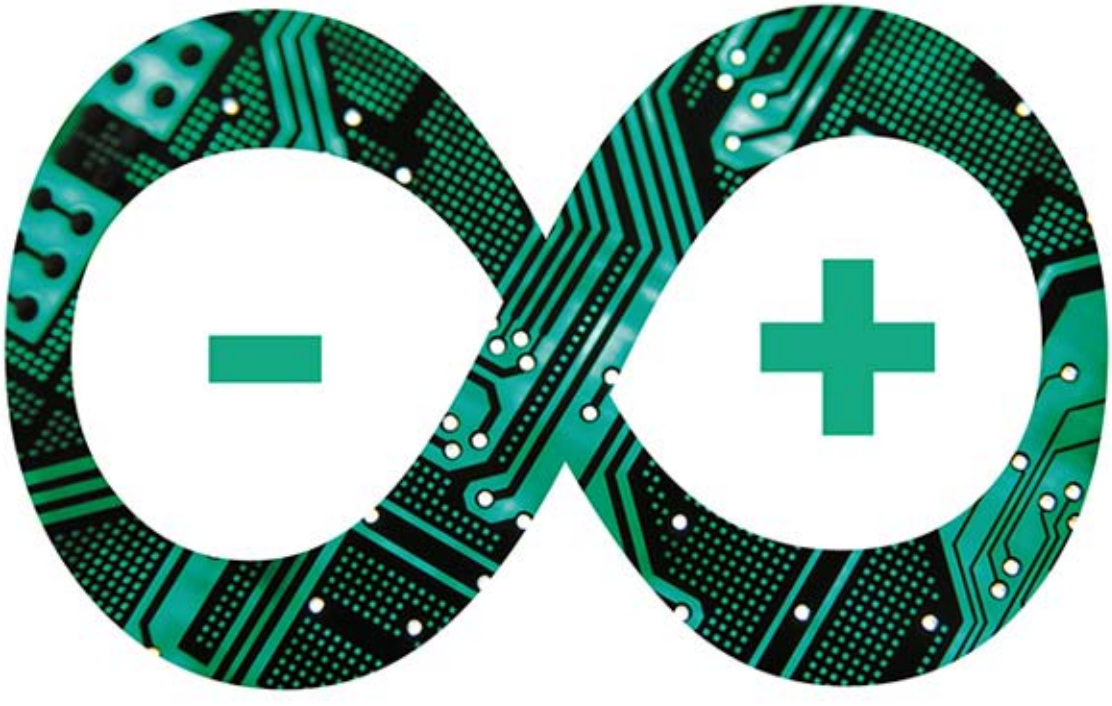
Martin Evans, Joshua Noble, Jordan Hochenbaum

ARDUINO W AKCJI

POZNAJ MOŻLIWOŚCI PLATFORMY ARDUINO!



Helion

Tytuł oryginału: Arduino in Action

Tłumaczenie: Jacek Janczyk (wstęp, rozdz. 1 – 4), Andrzej Watrak (rozdz. 5 - 13, dodatki)
Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
ISBN: 978-83-246-6356-9

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.
Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/arduak.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/arduak
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.
Printed in Poland.

Spis treści

8 Komunikacja

Rozdział ten omawia:

- konfigurację serwera WWW udostępniającego dane z Arduino,
- przesyłanie komunikatów z Arduino do portalu Twitter,
- komunikację z Arduino za pomocą WiFi i Bluetooth,
- zapisywanie danych na karcie SD i w sieci Internet w serwisie Xively,
- komunikację z innymi urządzeniami za pomocą protokołu SPI (ang. *Serial Peripheral Interface*, szeregowy interfejs urządzeń peryferyjnych).

W poprzednim rozdziale dowiedziałeś się, jak otrzymywać za pomocą wyświetlaczy LCD wizualną informację zwrotną z Arduino. Wyobraź sobie możliwość prezentowania informacji z Arduino na zewnętrznym ekranie i przesyłanie jej przez Internet w szeroki świat! A gdyby jeszcze można było zdalnie sterować płytą Arduino?

Podłączenie Arduino do Internetu i zdalne wysyłanie danych do Twojego komputera to dwie z wielu możliwych form komunikacji dostępnych w Arduino. Przyjrzymy się komunikacji wykorzystującej technologie Ethernet, WiFi, Bluetooth i SPI.

Ponieważ wiele Twoich projektów będzie wykorzystywać komunikację przez Internet, zajmijmy się nią od razu i sprawdźmy, jak Arduino może przesyłać dane przez sieć komputerową.

8.1. Technologia Ethernet

Jedną z najbardziej użytecznych form komunikacji dostępnych w Arduino jest technologia Ethernet. Jest to przyjęty standard budowy sieci komputerowych, umożliwiający urządzeniom wszelkiego rodzaju komunikację między sobą poprzez wysyłanie i odbieranie strumieni danych (zwanych **pakietami** lub **ramkami**).

Technologia Ethernet umożliwia wyjątkowo szybką i niezawodną transmisję danych przez sieć. Każde urządzenie posiada unikalny identyfikator zwany adresem IP, umożliwiający komunikację przy użyciu różnych protokołów sieciowych.

Komunikacja Arduino z siecią Internet jest prosta dzięki nakładce i bibliotece Ethernet, ale zanim omówimy te komponenty, poznajmy kilka pojęć związanych z sieciami komputerowymi. Nawet jeżeli już je znasz, warto przypomnieć sobie terminologię i technologie opisane w tabeli 8.1.

Tabela 8.1. Najważniejsze terminy i pojęcia technologii Ethernet

Termin	Opis
Ethernet	Ethernet jest standardową technologią wykorzystywaną do budowy sieci komputerowych, opisującą sposób przesyłania danych pomiędzy komputerami lub innymi urządzeniami przez sieć przewodową.
Protokół	Protokoły są to przyjęte języki komunikacji umożliwiające urządzeniom porozumiewanie się między sobą. Aby dwa urządzenia mogły się ze sobą komunikować, muszą używać tego samego języka. Na przykład protokół HTTP (ang. *Hypertext Transfer Protocol*, protokół przesyłania dokumentów hipertekstowych) jest powszechnie stosowanym protokołem, który możesz wykorzystać w płycie Arduino skonfigurowanej jako serwer WWW. Protokół HTTP określa język, dzięki któremu serwer WWW Arduino rozumie komunikaty i zapytania odbierane od innych systemów, na przykład komputerów z przeglądarkami.
Adres MAC	Adres MAC (ang. *Media Access Control*, kontrola dostępu do medium) jest to unikatowy identyfikator przypisywany urządzeniom wykorzystującym Ethernet lub inną technologię sieciową. Adres MAC umożliwia jednoznaczne zidentyfikowanie urządzenia w sieci, aby mogło komunikować się z innymi. Nakładka Ethernet Arduino posiada etykietę z przypisanym adresem MAC.
TCP/IP	Protokoły TCP (ang. *Transmission Control Protocol*, protokół sterowania transmisją danych) oraz IP (ang. *Internet Protocol*, protokół internetowy) umożliwiają przesyłanie komunikatów przez globalną sieć Internet (będącą fundamentem znanej i lubianej przez nas sieci WWW).
Adres IP	Adres IP jest to unikatowy identyfikator wykorzystywany przez urządzenia i serwery do identyfikacji w globalnej sieci Internet. Na przykład przy otwieraniu strony internetowej *www.helion.pl* usługa DNS (ang. *Directory Name Service*, usługa słownika *nazw*) zamienia adres *http://www.helion.pl* na numeryczny adres IP *188.117.147.100*.
Lokalny adres IP	Lokalny adres IP jest podobny do zwykłego adresu IP, ale jest używany w szczególności do komunikacji pomiędzy komputerami i urządzeniami w lokalnej sieci. Na przykład w Twojej domowej sieci każdy komputer ma przypisany lokalny adres IP, używany do komunikacji z routerem i innymi komputerami.

Sieci komputerowe i technologia Ethernet są złożonymi zagadnieniami, których pełne poznanie może zająć całe lata, ale w tabeli 8.1 wymienione są najbardziej podstawowe pojęcia, które musisz znać, aby zrozumieć pozostałą część tego rozdziału.

Teraz, po zapoznaniu się z tabelą, przejdźmy do biblioteki *Ethernet* i sprawdźmy, do czego służy.

8.1.1. Biblioteka Ethernet

Biblioteka *Ethernet* jest dostarczana razem ze środowiskiem Arduino IDE. Umożliwia ona konfigurację nakładki Ethernet i komunikację ze światem zewnętrznym oraz skonfigurowanie do czterech (w sumie) równolegle działających serwerów i klientów. Serwer przyjmuje połączenia przychodzące od klientów, po czym wysyła i odbiera dane. Natomiast klient najpierw zestawia z serwerem połączenie wychodzące, dzięki któremu może wysyłać i odbierać od niego dane.

Więcej na ten temat powiemy wkrótce, ale teraz spójrzmy na tabelę 8.2, zawierającą przegląd funkcji dostępnych w bibliotece *Ethernet*.

Tabela 8.2. Przegląd funkcji klas Ethernet, Server i Client dostępnych w bibliotece Ethernet

Funkcja	Opis

`Ethernet.begin(`*mac*`)`
`Ethernet.begin(`*mac*`, `*ip*`)`
`Ethernet.begin(`*mac*`, `*ip*`, `*brama*`)`
`Ethernet.begin(`*mac*`, `*ip*`, `*brama*`, `
`↪`*podsieć*`)`
	Inicjuje bibliotekę za pomocą adresu MAC nakładki i automatycznie konfiguruje adres IP za pomocą serwera DHCP. Opcjonalnie można ręcznie podać adres IP, bramę (najczęściej jest to adres IP routera) i maskę podsieci (informującą nakładkę, jak interpretować adres IP; domyślna maska to 255.255.255.0).

`Server(`*port*`)`	Tworzy serwer nasłuchujący na określonym porcie.
`Server.begin()`	Uruchamia serwer oczekujący na komunikaty.
`Server.available()`	Zwraca obiekt klienta, jeżeli są już odebrane od niego dane.
`Server.write()`	Wysyła dane do wszystkich podłączonych klientów.
`Server.print()`	Wysyła dane do wszystkich klientów. Liczby są wysyłane jako ciągi znaków ASCII, na przykład liczba *123* jest wysyłana jako ciąg trzech znaków: *'1'*, *'2'* i *'3'*.
`Server.println()`	Działa podobnie jak `Server.print()`, ale dodatkowo wysyła znak nowego wiersza na końcu każdego komunikatu.
`Client(`*ip*`, `*port*`)`	Tworzy obiekt klienta, który może łączyć się z określonym adresem IP i portem.
`Client.connected()`	Zwraca informację, czy klient jest połączony z serwerem. Jeżeli połączenie jest zamknięte, a jakieś dane wciąż nie są odczytane, funkcja zwraca wartość `true`.
`Client.connect()`	Nawiązuje połączenie.
`Client.write()`	Wysyła dane do serwera.
`Client.print()`	Wysyła dane do serwera. Liczby są wysyłane jako ciągi znaków ASCII, na przykład liczba *123* jest wysyłana jako ciąg trzech znaków: *'1'*, *'2'* i *'3'*.
`Client.println()`	Działa podobnie jak `Client.print()`, ale dodatkowo wysyła znak nowego wiersza na końcu każdego komunikatu.
`Client.available()`	Zwraca liczbę bajtów gotowych do odczytania (liczbę bajtów wysłanych przez serwer).
`Client.read()`	Odczytuje następny bajt odebrany z serwera.
`Client.flush()`	Usuwa wszystkie bajty wysłane do klienta, ale jeszcze przez niego nieodczytane.
`Client.stop()`	Zamyka połączenie z serwerem.

Oprócz klas `Ethernet`, `Server` oraz `Client` biblioteka *Ethernet* zawiera przydatne klasy ogólnego przeznaczenia, obsługujące protokół *UDP* (ang. *User Datagram Protocol*,

protokół danych użytkownika) do rozgłaszania informacji w sieci. Jeżeli serwer lub klient nie jest wymagany, możesz do wysyłania i odbierania danych z Arduino użyć klasy `EthernetUDP`. Tabela 8.3 zawiera szczegółowy opis funkcji tej klasy.

Tabela 8.3. Przegląd głównych funkcji klasy EthernetUDP w bibliotece Ethernet

Funkcja	Opis
`EthernetUDP.begin(port)`	Inicjuje obiekt `UDP` i określa port do nasłuchu.
`EthernetUDP.read(bufor_pakietów, maks_wielkość)`	Odczytuje z bufora pakiety UDP.
`EthernetUDP.write(komunikat)`	Wysyła komunikat do innego urządzenia.
`EthernetUDP.beginPacket(ip, port)`	Określa adres IP i port urządzenia docelowego. Funkcja musi być wywołana przed wysłaniem komunikatu.
`EthernetUDP.endPacket()`	Kończy komunikat. Funkcja wywoływana po wysłaniu komunikatu.
`EthernetUDP.parsePacket()`	Sprawdza, czy jakiś komunikat oczekuje na odczytanie.
`EthernetUDP.available()`	Zwraca ilość danych odebranych i gotowych do odczytania.

Teraz, kiedy znamy już funkcje i klasy biblioteki *Ethernet*, przyjrzyjmy się samej nakładce. Nakładkę stosuje się w celu rozszerzenia funkcjonalności sprzętowych Arduino. Dzięki niej możesz podłączyć płytę do sieci Ethernet.

8.1.2. Nakładka Ethernet z kartą SD

Oryginalna nakładka Ethernet jest kamieniem milowym w rozwoju platformy Arduino. Umożliwia ona komunikację projektów z innymi urządzeniami przez sieć komputerową lub Internet. Nakładka wykorzystuje układ WIZnet W5100 i oferuje obsługę stosu IP wraz z protokołami TCP i UDP poprzez łącza Ethernet 10/100 Mbit/s. Nakładka jest wyposażona w standardowe gniazdo RJ45 umożliwiające połączenie i współpracę z modemem, routerem lub innymi standardowymi urządzeniami.

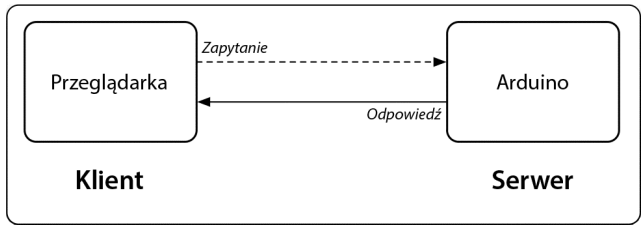
Nowsza, szeroko dostępna wersja nakładki zawiera ulepszenie w postaci slotu na kartę microSD. Dzięki niemu można odczytywać i zapisywać pliki (za pomocą biblioteki *SD*, po umieszczeniu karty SD w slocie). Należy pamiętać, że zarówno układ W5100, jak i karta SD komunikują się z Arduino za pomocą interfejsu SPI (więcej informacji na temat komunikacji SPI znajduje się w podrozdziale 8.6). W większości płyt Arduino wykorzystywane są w tym celu piny nr 11, 12 i 13, natomiast w płycie Mega piny 50, 51 i 52. W obu płytach pin nr 10 jest używany do wybrania do komunikacji układu W5100, natomiast pin nr 4 do wybrania karty SD. Jest to istotne z tego względu, że nie można tych pinów używać jako wejściowych lub wyjściowych pinów ogólnego przeznaczenia.

> **UWAGA:** Zarówno układ W5100, jak i karta SD korzystają z szyny SPI, ale tylko jedno z nich może być aktywne w danej chwili. Aby móc korzystać z karty SD, należy skonfigurować pin nr 4 jako wyjście i ustawić na nim stan wysoki (`HIGH`). Natomiast w przypadku układu W5100 należy jako wyjście skonfigurować pin nr 10 i ustawić na nim stan wysoki. Sprzętowego pinu SS (w większości płyt jest

to pin nr 10, a w płycie Mega nr 53) można nie używać, ale aby móc korzystać z karty SD, biblioteki *Ethernet* i interfejsu SPI, trzeba ustawić go jako wyjście (domyślna konfiguracja).

8.2. Serwer WWW Arduino

Mając w małym palcu podstawy technologii, bibliotekę i nakładkę, możesz zacząć pracę nad swoim pierwszym projektem wykorzystującym Ethernet. Zbudujesz **serwer WWW**, czyli system odpowiadający na zapytania wysyłane przez **klientów** i wysyłający do nich dane (jak przedstawia rysunek 8.1). Aby to osiągnąć, wykorzystasz klasy `Server` oraz `Client` z biblioteki *Ethernet*.



Rysunek 8.1. Schemat komunikacji klienta z serwerem WWW Arduino

8.2.1. Konfiguracja serwera

Do skonfigurowania serwera będziesz potrzebować pewnych informacji.

Po pierwsze, musisz znać adres MAC swojej nakładki Ethernet. Powinien on być wydrukowany na etykiecie na nakładce. W kodzie serwera adres MAC zapiszesz w tabeli bajtów, na przykład tak:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```

Pamiętaj, że jest to unikatowy adres sprzętowy, wykorzystywany przez nakładkę do bezpośredniej komunikacji z innymi urządzeniami przez sieć Ethernet.

Jeżeli posiadasz starą nakładkę bez etykiety albo ją zgubiłeś, możesz wykorzystać adres MAC z powyższego przykładu. Jeżeli do sieci jest dołączonych więcej niż jedno urządzenie, ważne jest, aby każde posiadało swój własny unikatowy adres MAC.

W następnym kroku będziesz potrzebować adresu IP. Od wersji 1.0 Arduino wbudowana biblioteka *Ethernet* obsługuje protokół *DHCP* (*Dynamic Host Configuration Protocol*, protokół dynamicznego konfigurowania urządzenia), umożliwiający Arduino automatyczne przypisanie adresu IP. Adres IP zostanie automatycznie skonfigurowany niezależnie od tego, czy płyta jest podłączona bezpośrednio do modemu, czy do routera, o ile tylko na tych urządzeniach jest włączona usługa DHCP (zazwyczaj jest).

Jeżeli w Twojej sieci nie jest używany protokół DHCP albo używasz wersji Arduino starszej niż 1.0, musisz dowiedzieć się, jaki adres IP ma router, i ręcznie przypisać nakładce Ethernet jakiś inny adres IP. W zależności od konfiguracji jest kilka sposobów wykonania tej operacji.

Jeżeli płyta Arduino jest podłączona bezpośrednio do modemu, jego adres IP jest przypisany przez operatora internetowego ISP. W takim przypadku najprostszym

sposobem poznania adresu IP jest podłączenie komputera do modemu i skorzystanie z jednej z wielu stron w Internecie rozpoznających Twój adres IP. (Proste wyszukanie frazy „adres IP" powinno pomóc w znalezieniu odpowiedniej strony, ewentualnie możesz skorzystać z *http://www.adres-ip.pl*.) Zwróć uwagę, że adres IP Twojego routera jest automatycznie przypisywany przez operatora ISP i może się zmieniać od czasu do czasu.

Jeżeli płyta Arduino jest dołączona do sieci przez router, musisz ręcznie przypisać do nakładki jakiś niewykorzystywany adres IP. Aby to zrobić, musisz posiadać nieco więcej informacji na temat konfiguracji swojej sieci. Adres IP routera możesz poznać, korzystając z komputera dołączonego do sieci i opisanej wyżej usługi internetowej. Możesz również otworzyć panel administracyjny swojego routera, wpisując w przeglądarce jego domyślny adres IP. Na przykład w przypadku urządzeń Linksys jest to adres *http://192.168.1.1*, a dla innych marek *http://10.0.0.1*. Jeżeli Twój router ma np. adres 192.168.1.1, musisz nakładce Ethernet nadać adres 192.168.1.x, gdzie *x* oznacza dowolną liczbę od 2 do 254. Każdy komputer lub inne urządzenie w sieci posiada swój unikatowy adres 192.168.1.x, w którym ostatnia liczba identyfikuje to urządzenie w sieci. Dlatego musisz się upewnić, że Twoja nakładka nie wchodzi w konflikt z innymi urządzeniami. Ta sama zasada obowiązuje, gdy Twój router wykorzystuje inną adresację, na przykład 10.0.0.x.

Jeżeli musisz ręcznie przypisać adres IP, możesz wykorzystać obiekt `IPAddress`, na przykład:

```
IPAddress manualIP(192, 168, 1, 2);
```

Liczby w powyższym kodzie oznaczają adres, który chcesz przypisać.

Na koniec jeżeli jesteś podłączony do sieci i wykorzystujesz router do połączenia z Internetem, musisz również ustawić jego adres IP jako bramę. Utwórz tabelę typu `byte` i zapisz w niej adres IP routera, który będzie użyty jako adres bramy. Poniżej przedstawiony jest przykład:

```
byte gateway[] = { 192, 168, 1, 1};
```

Teraz, kiedy znasz już konfigurację IP sieci, przejdźmy do kodu.

8.2.2. Szkic konfigurujący serwer WWW

Listing 8.1 stanowi praktyczne zastosowanie omówionych wcześniej zagadnień do skonfigurowania serwera WWW na płycie Arduino. Serwer będzie miał za zadanie dołączyć się do Internetu (lub do sieci lokalnej), przyjmować połączenia przychodzące od klientów (wysyłane przez przeglądarki) i odpowiadać własnym komunikatem. Listing jest doskonałym szablonem do tworzenia niemal dowolnej aplikacji serwerowej.

Listing 8.1. Serwer WWW na płycie Arduino

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```

Przypisanie unikatowego adresu MAC

```
IPAddress manualIP(192,168,1,120);
EthernetServer server(80);

boolean dhcpConnected = false;

void setup()
{
  if (!Ethernet.begin(mac)){
    Ethernet.begin(mac, manualIP);
  }
  server.begin();
}

void loop()
{
  EthernetClient client = server.available();
  if (client) {
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();

        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();
          client.println("Witaj, jestem Twoim serwerem Arduino!");
          break;
        }
        if (c == '\n') {
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          currentLineIsBlank = false;
        }
      }
    }
    delay(1);
    client.stop();
  }
}
```

Ręczne przypisanie adresu IP, jeżeli DHCP nie działa

❶ **Inicjalizacja serwera**

❷ **Połączenie za pomocą DHCP**

❸ **Ręczne połączenie, jeżeli DHCP nie działa**

❹ **Uruchomienie serwera**

Oczekiwanie na połączenia od klientów

Połączenie i odczyt danych od klienta

❺ **Koniec zapytania i odesłanie odpowiedzi do klienta**

Zwłoka na odczytanie danych przez klienta

Zamknięcie połączenia

Najpierw inicjowany jest serwer HTTP na porcie nr 80 ❶. Po zainicjowaniu możesz spróbować podłączyć Arduino do sieci Ethernet, korzystając z usługi DHCP ❷, a jeżeli nie będzie to możliwe, ręcznie przypisując adres ❸. Dalej następuje uruchomienie serwera i w głównej pętli rozpoczyna się nasłuchiwanie połączeń ❹.

Gdy do serwera podłączy się klient i odebrane zostaną dane, wówczas znak nowego wiersza będzie oznaczał koniec komunikatu, po czym do klienta zostanie odesłana odpowiedź ❺.

8.2.3. Załadowanie i test szkicu

Skopiuj dokładnie kod z listingu 8.1 do środowiska Arduino IDE. Teraz możesz załadować szkic do Arduino.

Po załadowaniu i uruchomieniu kodu możesz zdalnie połączyć się z serwerem Arduino, otwierając w dowolnej przeglądarce adres *http://twój_adres_arduino*. Twoja przeglądarka (klient) wyśle do serwera żądanie połączenia, który z kolei odpowie komunikatem „*Witaj, jestem Twoim serwerem Arduino!*". Tak to działa.

Chcesz otrzymywać rzeczywiste dane? Podłącz potencjometr lub czujnik do wejścia analogowego nr 0 i zamień wiersz:

```
client.println("Witaj, jestem Twoim serwerem Arduino!");
```

na następujący:

```
client.println(analogRead(0));
```

Mamy nadzieję, że szkic zadziała bez problemów, ale jeżeli nie otrzymasz żadnej odpowiedzi, przeczytaj poniższy punkt poświęcony usuwaniu usterek.

8.2.4. Usuwanie usterek

Jeżeli nie możesz nawiązać połączenia z Arduino, pierwszą rzeczą, którą należy sprawdzić, są ustawienia adresu IP. Jeżeli jesteś absolutnie pewny, że konfiguracja IP jest poprawna i jesteś podłączony do sieci domowej, możliwe, że musisz skonfigurować na routerze przekierowanie portów. Przekierowanie portów powoduje, że router w specjalny sposób przesyła komunikaty przeznaczone dla Twojego Arduino. Konfiguracja przekierowania portów nie jest trudna; musisz ją wykonać na swoim routerze. Aby uzyskać więcej informacji, zajrzyj do dokumentacji routera, jak ustawić przekierowanie na adres IP Arduino. To powinno rozwiązać problem.

8.3. Ćwir, ćwir — komunikacja z portalem Twitter

Tworzenie serwera WWW komunikującego się ze światem zewnętrznym to wspaniałe zajęcie, ale inną użyteczną opcją jest połączenie z usługami w Internecie. Jedną z usług, z której warto skorzystać, jest portal Twitter.

Zasada działania portalu Twitter jest prosta. Jeżeli masz w nim swoje konto, możesz rozsyłać w całej sieci Twitter tweety (komunikaty) o długości maksymalnie 140 znaków. Użytkownicy mogą zapisywać się do Twojego kanału i automatycznie otrzymywać aktualizacje Twoich tweetów. Ale to nie wszystko. Twitter dobrze współpracuje z innymi usługami, czyli możesz również automatycznie wysyłać swoje tweety do konta w portalu Facebook.

Czy nie byłoby wspaniale skonfigurować kanał Twitter dostarczający aktualnych informacji o tym, co się dzieje z Twoim Arduino? Jest to możliwe. W tym podrozdziale dowiesz się, jak skonfigurować Arduino i nakładkę Ethernet, aby po naciśnięciu przycisku podłączonego do płyty automatycznie wysyłać komunikaty do portalu Twitter.

8.3.1. Twitter i tokeny

Jeżeli nie masz jeszcze konta w portalu Twitter albo na potrzeby tego projektu chcesz założyć nowe, odwiedź stronę *www.twitter.com* i utwórz konto.

Następnie pobierz specjalny token, który umożliwi autoryzację Arduino podczas wysyłania komunikatów przez Twoje konto. Ten token umożliwia pośredniemu serwerowi WWW mediację pomiędzy Arduino a portalem Twitter. Możliwa jest również bezpośrednia komunikacja z portalem, ale korzystanie z usługi pośredniczącej jest lepsze, ponieważ zapobiega przerwaniu wykonywania Twojego kodu w przypadku, gdy Twitter zmieni swój protokół komunikacyjny lub sposób autoryzacji. Również biblioteka *Twitter* jest mniejsza dzięki usłudze pośredniczącej. Oszczędza się w ten sposób cenną pamięć Arduino.

Aby pobrać token, otwórz stronę *http://arduino-tweet.appspot.com* i kliknij odnośnik *Step 1: Get a token to post a message using OAuth* (Krok 1. Pobierz token do wysyłania komunikatów za pomocą usługi OAuth).

Po skonfigurowaniu konta pora przyjrzeć się bliżej bibliotece, z której będziemy korzystać.

8.3.2. Biblioteki i funkcje

Aby móc komunikować się z portalem Twitter, musisz zainstalować bibliotekę *Twitter*, dostępną pod adresem *www.arduino.cc/playground/Code/TwitterLibrary*. Po pobraniu biblioteki zapisz ją w folderze *sketchbook* lub *libraries*.

Tabela 8.4 przedstawia opis funkcji zawartych w bibliotece *Twitter*.

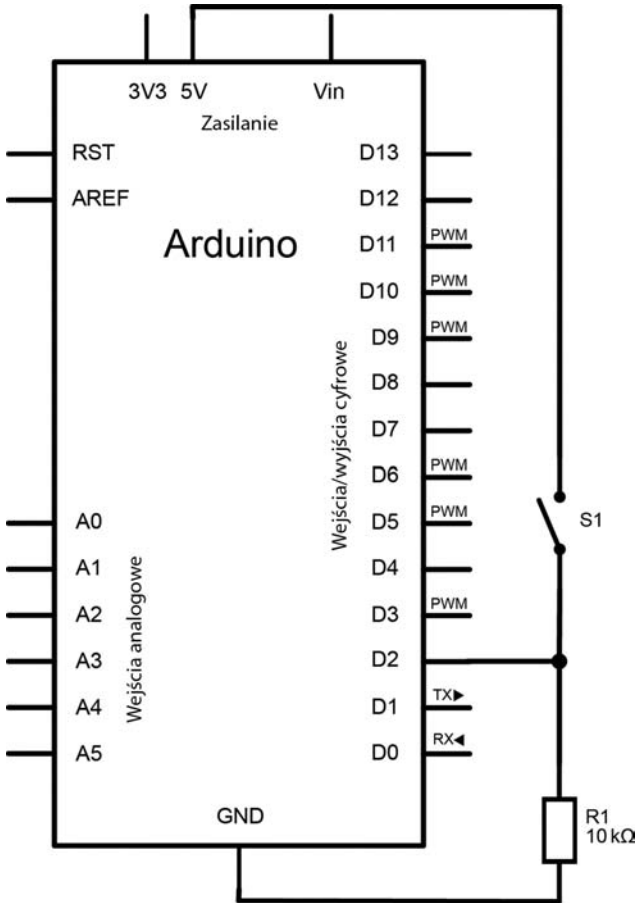
Tabela 8.4. Przegląd funkcji biblioteki Twitter

Funkcja	Opis
`Twitter(string token)`	Konstruktor klasy, przyjmujący token jako argument.
`bool post(const char *komunikat)`	Rozpoczyna wysyłanie komunikatu. Zwraca wartość `true` po pomyślnym nawiązaniu połączenia z portalem Twitter lub `false` w przypadku błędu.
`bool checkStatus(Print *debug)`	Sprawdza, czy żądanie opublikowania komunikatu jest wciąż realizowane (można pominąć argument `debug`, jeżeli nie jest wymagana informacja zwrotna).
`int status()`	Zwraca kod HTTP odpowiedzi z portalu Twitter, na przykład *200 OK* Status jest dostępny dopiero po opublikowaniu komunikatu, gdy funkcja `checkStatus()` zwróci wartość `false`.
`int wait(Print *debug)`	Czeka na zakończenie publikacji komunikatu. Zwraca kod HTTP odpowiedzi z portalu Twitter.

8.3.3. Schemat układu i połączenia komponentów

Jeżeli konto w portalu Twitter i środowisko Arduino IDE są skonfigurowane, zbudujmy prosty układ wysyłający tweeta, gdy użytkownik naciśnie przycisk. Przylutuj przycisk do płyty lub zbuduj układ na płycie prototypowej.

Podłącz jedną z końcówek przycisku z pinem 5 V, a drugą z pinem masy GND poprzez rezystor obniżający 100 kΩ. Tę samą końcówkę przycisku (podłączoną do masy) dołącz do wejścia cyfrowego nr 2, jak pokazuje rysunek 8.2.



Rysunek 8.2. Prosty układ do wysyłania tweetów po naciśnięciu przycisku

8.3.4. Szkic do wysyłania tweeta po naciśnięciu przycisku

Po podłączeniu przycisku i umieszczeniu na Arduino nakładki Ethernet skopiuj poniższy kod (listing 8.2) do środowiska Arduino IDE i możesz zaczynać. Przeczytaj uważnie komentarze w kodzie i upewnij się, że kod zawiera ustawienia odpowiadające konfiguracji Twojej sieci. Jeżeli potrzebujesz wyjaśnień dotyczących jakiegoś pojęcia sieciowego lub terminu, wróć do podrozdziału 8.1.

Listing 8.2. Szkic do wysyłania komunikatu do portalu Twitter po naciśnięciu przycisku

```
#include <SPI.h>
#include <Ethernet.h>
#include <Twitter.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress manualIP(192,168,1,120);

int b1Pin = 2;
int pressCount = 0;
```

Przypisanie unikatowego adresu MAC

Ręczne przypisanie adresu IP, jeżeli DHCP nie działa

Określenie pinu przycisku

```
Twitter twitter("TWOJ-TOKEN");

void setup()
{
  delay(1000);
  if(!Ethernet.begin(mac)){
    Ethernet.begin(mac, manualIP);
  }
  Serial.begin(9600);
}

void sendTweet(const char msgToSend[])
{
  Serial.println("Laczenie...");
  if (twitter.post(msgToSend)) {
    int status = twitter.wait(&Serial);
    if (status == 200) {
      Serial.println("OK.");
    }
    else {
      Serial.print("Blad : kod ");
      Serial.println(status);
    }
  }
  else {
    Serial.println("Polaczenie nieudane.");
  }
}

void loop()
{
  if(digitalRead(b1Pin) == HIGH)
  {
    pressCount++;
    sendTweet("Liczba nacisniec przycisku: " + pressCount);
    delay(2000);
  }
}
```

Inicjalizacja tokena do portalu Twitter

❶ **Połączenie za pomocą DHCP**

❷ **Ręczne połączenie, jeżeli DHCP nie działa**

❸ **Zestawienie połączenia szeregowego**

❹ **Sformatowanie i wysłanie tweeta**

❺ **Sprawdzenie przycisku i wysłanie tweeta**

Najpierw Arduino próbuje podłączyć się do sieci Ethernet, korzystając z usługi DHCP ❶, a jeżeli nie będzie to możliwe, za pomocą ręcznie przypisanego adresu IP ❷. Po podłączeniu otwierane jest połączenie szeregowe, poprzez które wysyłane są komunikaty diagnostyczne do monitora portu szeregowego ❸.

Jeżeli połączenie z portalem zostanie nawiązane, zastosowana zostanie funkcja `sendTweet`, odpowiednio formatująca i wysyłająca tweeta ❹. Kod sprawdza, czy został naciśnięty przycisk, po czym wysyła tweeta ❺.

8.3.5. Załadowanie i test szkicu

Jeżeli jesteś pewien, że szkic zawiera poprawne dane, skompiluj go i załaduj do Arduino. Teraz możesz rozgłaszać naciśnięcia przycisku w portalu Twitter. Sprawdź, czy płyta Arduino wykonuje kod i czy jest podłączona do Internetu przewodem Ethernet. To jest cała konfiguracja.

Naciśnij przycisk podłączony do Arduino i zaloguj się do portalu Twitter. Powinieneś w nim zobaczyć ostatni tweet o treści „Liczba nacisniec przycisku: 1", jak pokazuje rysunek 8.3. Po każdym naciśnięciu przycisku pojawi się nowy tweet ze zwiększającą się liczbą naciśnięć. Fajna zabawa.

Rysunek 8.3. Widok tweetu w portalu Twitter po naciśnięciu przycisku

> **UWAGA:** Twitter blokuje wielokrotne wysyłanie w krótkim przedziale czasu tego samego komunikatu. W listingu 8.2 komunikat jest zmieniany po każdym naciśnięciu przycisku (zwiększana jest liczba naciśnięć), a więc nie będzie problemu z wysłaniem tweeta. W większości przypadków okresowego wysyłania komunikatów lub komunikatów różniących się między sobą ten problem nie powinien wystąpić. Niemniej jednak warto o tym pamiętać.

Jak zapewne się domyślasz, możesz rozgłaszać wiele innych pożytecznych informacji, nie tylko dotyczących naciśnięcia przycisku. Na przykład czujnik może wysyłać sygnał, gdy drzwi do Twojego domu zostaną otwarte lub zamknięte. Możesz wysyłać bieżące dane i przedstawiać je w swojej galerii. To tylko zarys możliwości.

8.4. Łączność Wi-Fi

Nakładka Ethernet błyskawicznie podłączy Twoją płytę Arduino do sieci, ale w niektórych sytuacjach przydaje się łączność bezprzewodowa. Na przykład gdy musisz odbierać na bieżąco dane z samobieżnego robota własnej konstrukcji albo gdy Twój układ musi być podłączony do sieci, a w pobliżu nie ma łącza ani routera przewodowego. Sięgnij wtedy po nakładkę WiFi, eleganckie rozwiązanie umożliwiające podłączenie Arduino do sieci bezprzewodowej i przesyłanie danych.

> **UWAGA:** Jeżeli posiadasz nakładkę SparkFun WiFly (popularny odpowiednik oficjalnej nakładki Arduino WiFi) albo z jakiegoś powodu musisz ją wykorzystać w projekcie, na stronie internetowej oryginalnego wydania książki znajdziesz odpowiednio zmienioną wersję tego podrozdziału (w języku angielskim). Odnośnik do niego zamieściliśmy w dodatku E.

8.4.1. Nakładka Arduino WiFi

Nakładka Arduino WiFi umożliwia podłączenie płyty do dowolnej sieci bezprzewodowej typu 802.11b/g. Wykorzystuje moduł bezprzewodowy H&D Wireless HDG104, oferujący uzyskanie zoptymalizowanego, energooszczędnego połączenia radiowego. Nakładka umożliwia komunikację za pomocą protokołów TCP i UDP, a jej użycie jest bardzo proste i polega na zamontowaniu na płycie Arduino i wpisaniu w szkicu kilku wierszy kodu wykorzystującego bibliotekę *WiFi*. Łączówki nakładki posiadają w górnej części gniazda, umożliwiające łatwe wykorzystanie pinów Arduino albo założenie dodatkowych nakładek.

Oprócz obsługi sieci bezprzewodowych w standardzie 802.11b/g nakładka WiFi oferuje szyfrowanie WEP oraz WPA2. Po załadowaniu szkicu i skonfigurowaniu płyty Arduino można ją odłączyć od komputera, zasilić z zewnętrznego źródła i zestawić dwukierunkową komunikację z dowolnego miejsca w zasięgu routera bezprzewodowego.

Ale to nie wszystko. Nakładka WiFi zawiera również slot na kartę microSD, który może być wykorzystany zarówno przez płytę Arduino Uno, jak i Mega, dzięki prostej w użyciu bibliotece *SD*. Jest to bardzo przydatna funkcjonalność, jeżeli zamierzasz zapisać dane, a następnie przesłać je przez sieć. W podrozdziale 8.7 dowiesz się dokładnie, jak korzystać z biblioteki *SD*.

Ważna informacja na temat pinów wejścia/wyjścia

Nakładka Arduino WiFi oraz czytnik kart SD komunikują się z Arduino za pomocą szyny SPI (opisanej niżej w podrozdziale 8.6), którą cechuje kilka istotnych szczegółów związanych z wykorzystaniem pinów wejścia/wyjścia.

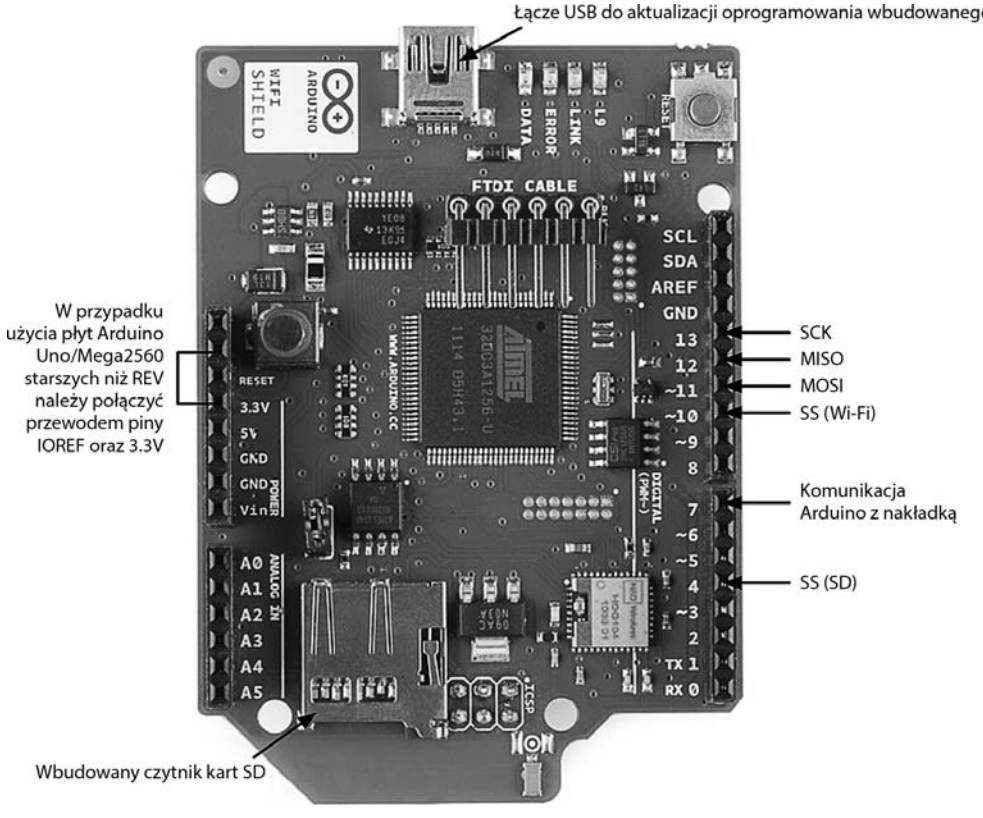
W płycie Arduino Uno komunikacja jest realizowana na pinach nr 11, 12 i 13, natomiast w płycie Mega na pinach nr 50, 51 i 52. W obu płytach pin nr 10 jest używany do wybrania do komunikacji układu HDG104, natomiast pin nr 4 do wybrania czytnika karty SD. Sprzętowy pin SS w płycie Mega (pin cyfrowy nr 53) nie jest używany ani przez czytnik kart, ani przez układ HDG104, ale musi być skonfigurowany jako wyjście, aby interfejs SPI działał prawidłowo. Pin cyfrowy nr 7 jest używany do przesyłania danych pomiędzy nakładką WiFi a Arduino. Bardzo ważne jest więc, aby żaden z wymienionych wyżej pinów nie był wykorzystywany do innych operacji wejścia/ wyjścia.

I wreszcie ponieważ zarówno układ HDG104 w nakładce WiFi, jak również czytnik kart SD korzystają z tej samej szyny SPI, tylko jeden z komponentów może być aktywny w danej chwili. Jeżeli wykorzystywane są oba, biblioteki *SD* oraz *WiFi* realizują ich obsługę automatycznie. Ale jeżeli jest wykorzystywany tylko jeden z nich, należy jawnie odseparować drugi (jeżeli nie jest używany czytnik kard SD, należy go ręcznie odseparować), jak pokazuje przykładowy kod.

Konstrukcja nakładki WiFi jest starannie przemyślana i oprócz nawiązywania połączeń bezprzewodowych oferuje szereg przydatnych funkcjonalności. Jej budowa jest całkowicie otwarta. Nakładka jest wyposażona w port Micro-USB na potrzeby przyszłych aktualizacji wbudowanego oprogramowania. Zawiera również serię diod LED dostarczających przydatnych informacji, takich jak status połączenia (zielona dioda LINK), błędy transmisji (czerwona dioda ERROR) i wysyłanie lub odbieranie danych (niebieska dioda DATA).

JAK UŻYWAĆ NAKŁADKI WIFI ZE STARSZYMI PŁYTAMI ARDUINO?

Nakładka WiFi wykorzystuje pin IOREF, dostępny w nowszych wersjach płyty Arduino, do wykrywania napięcia odniesienia dla pinów wejścia/wyjścia, do których jest podłączona. Oznacza to, że w przypadku zastosowania płyty Arduino Uno lub Mega2560 w wersji wcześniejszej niż REV3 trzeba **koniecznie** zewrzeć piny nakładki IOREF oraz 3.3V (pokazane na rysunku 8.4).



Rysunek 8.4. Układ pinów nakładki WiFi

8.4.2. Biblioteka WiFi i jej funkcje

Biblioteka *WiFi* realizuje niskopoziomową komunikację bezprzewodową i obsługuje wiele poleceń i funkcjonalności oferowanych przez nakładkę.

W tym miejscu warto zapoznać się z tabelą 8.5, zawierającą przegląd najważniejszych funkcji z biblioteki *WiFi*.

Po przejrzeniu tabeli 8.5 możesz przejść do opisu przykładowego projektu, w którym przez sieć bezprzewodową będą przesyłane dane z czujnika gestów.

Tabela 8.5. Przegląd funkcji klas WiFi, WiFiServer oraz WiFiClient

Funkcja	Opis

`WiFi.begin()`
`WiFi.begin(char[] ssid)`
`WiFi.begin(char[] ssid, ↪char[] haslo)`
`WiFi.begin(char[] ssid, int iindeksKlucza, char[] klucz)`
	Inicjuje bibliotekę *WiFi* i rozpoczyna komunikację z urządzeniem. Umożliwia podłączenie do dowolnej otwartej sieci, jak również do sieci zabezpieczonej z szyfrowaniem WPA po podaniu identyfikatora SSID i hasła oraz sieci z szyfrowaniem WEP po podaniu indeksu i klucza (w szyfrowaniu WEP mogą być zastosowane cztery klucze, dlatego trzeba podać jego indeks). Funkcja zwraca status połączenia z siecią WiFi.

`WiFi.disconnect()`	Odłącza się od bieżącej sieci.
`WiFi.SSID()`	Odczytuje identyfikator SSID bieżącej sieci i zwraca go jako ciąg znaków typu `String`.
`WiFi.BSSID(bssid)`	Odczytuje adres MAC routera, z którym jest nawiązane połączenie, i umieszcza go w 6-bajtowej tabeli przekazanej jako argument (na przykład `byte bssid[6]`).
`WiFi.RSSI()`	Zwraca siłę sygnału połączenia jako liczbę typu `Long`.
`WiFi.encryptionType()` `WiFi.encryptionType ↪(wifiAccessPoint)`	Zwraca rodzaj szyfrowania bieżącego (lub wskazanego) punktu dostępowego. Zwracana wartość jest typu `byte`. W przypadku szyfrowania TKIP (WPA) = `2`, WEP = `5`, CCMP (WPA) = `4`, NONE = `7`, AUTO = `8`.
`WiFi.scanNetworks()`	Zwraca wartość typu `byte` zawierającą liczbę wykrytych sieci bezprzewodowych.
`WiFi.getSocket()`	Zwraca pierwsze dostępne gniazdo połączenia.
`WiFi.macAddress()`	Zwraca 6-bajtową tabelę zawierającą adres MAC nakładki WiFi.
`WiFi.localIP()`	Zwraca adres IP nakładki (jako obiekt typu `IPAddress`).
`WiFi.subnetMask()`	Zwraca maskę podsieci nakładki (jako obiekt typu `IPAddress`).
`WiFi.gatewayIP()`	Zwraca adres IP bramy (jako obiekt typu `IPAddress`).
`WiFiServer(int port)`	Tworzy serwer nasłuchujący na zadanym porcie.
`WiFiServer.begin()`	Uruchamia serwer oczekujący na komunikaty.
`WiFiServer.available()`	Zwraca obiekt klienta, jeżeli są już odebrane od niego dane.
`WiFiServer.write(data)`	Wysyła dane (typu `byte` lub `char`) do wszystkich dołączonych klientów.
`WiFiServer.print()`	Wysyła dane do wszystkich klientów. Liczby są wysyłane jako ciągi znaków ASCII, na przykład liczba *123* jest wysyłana jako ciąg trzech znaków: *'1'*, *'2'* i *'3'*.
`WiFiServer.println()`	Działa podobnie jak `WiFiServer.print()`, ale dodatkowo wysyła znak nowego wiersza na końcu każdego komunikatu.
`WiFiClient()`	Tworzy obiekt klienta, który może łączyć się z określonym adresem IP i portem określonym w funkcji `connect()`.
`WiFiClient.connected()`	Zwraca informację, czy klient jest połączony z serwerem. Jeżeli połączenie jest zamknięte, a jakieś dane wciąż nie są odczytane, funkcja zwróci wartość `true`.
`WiFiClient.connect(ip, port)` `WiFiClient.connect(URL, port)`	Nawiązuje połączenie z określonym adresem IP i portem. Adres URL jest zamieniany na adres IP.
`WiFiClient.write(data)`	Wysyła dane (typu `byte` lub `char`) do serwera.
`WiFiClient.print()`	Wysyła dane do klienta. Liczby są wysyłane jako ciągi znaków ASCII, na przykład liczba 123 jest wysyłana jako ciąg trzech znaków: *'1'*, *'2'* i *'3'*.

Tabela 8.5. Przegląd funkcji klas WiFi, WiFiServer oraz WiFiClient — *ciąg dalszy*

Funkcja	Opis
`WiFiClient.println()`	Działa podobnie jak `WiFiClient.print()`, ale dodatkowo wysyła znak nowego wiersza na końcu każdego komunikatu.
`WiFiClient.available()`	Zwraca liczbę bajtów gotowych do odczytania (liczbę bajtów wysłanych przez serwer).
`WiFiClient.read()`	Odczytuje następny bajt odebrany z serwera.
`WiFiClient.flush()`	Usuwa wszystkie bajty wysłane do klienta, ale jeszcze przez niego nieodczytane.
`WiFiClient.stop()`	Zamyka połączenie z serwerem.

8.4.3. Ruchy ciała i bezprzewodowe przyspieszeniomierze

W tym przykładzie zastosujesz Arduino z funkcją WiFi do bezprzewodowego przesyłania danych z przyspieszeniomierza. Przyspieszeniomierze to czujniki znakomicie nadające się do realizacji wszelkiego rodzaju interakcji za pomocą ruchów ciała. Można dzięki nim testować nowe scenariusze gier (jak w popularnych konsolach Nintendo Wii) lub po umieszczeniu ich na ciele tancerza wykorzystywać jego ruchy do sterowania efektami wizualnymi i dźwiękowymi. Przyspieszeniomierzy można również używać do wspomagania osób niepełnosprawnych fizycznie. Jak widzisz, jest mnóstwo praktycznych zastosowań przyspieszeniomierzy i na pewno przychodzą Ci do głowy kolejne.

W tym przykładzie będą Ci potrzebne:

- płyta Arduino,
- nakładka Arduino WiFi,
- przynajmniej jeden przyspieszeniomierz.

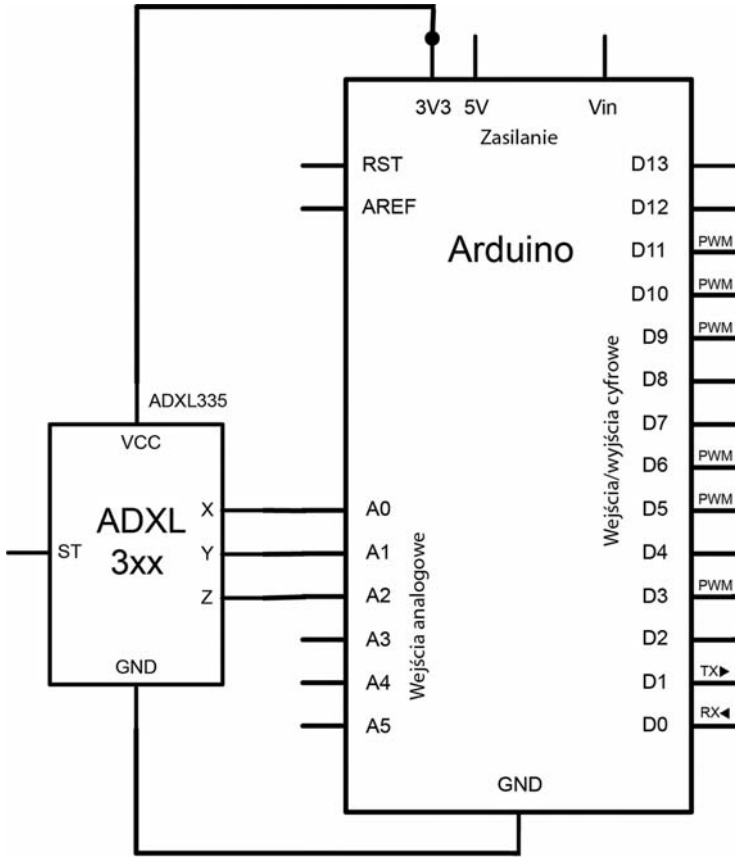
W przykładzie użyjesz języka Processing do utworzenia serwera, za pomocą którego będziesz przesyłać i analizować dane z bezprzewodowych przyspieszeniomierzy.

8.4.4. Łączenie komponentów

Podłączenie nakładki WiFi jest bardzo proste i polega na założeniu jej bezpośrednio na płytę Arduino od jej wierzchniej strony. Sposób podłączenia przyspieszeniomierza jest różny i zależy od zastosowanego modelu.

Jeżeli korzystasz z trójosiowego przyspieszeniomierza ADXL335 posiadającego niezależne wyjścia analogowe dla każdej osi (x, y, z), połącz go zgodnie z rysunkiem 8.5. Jeżeli posiadasz inny model, połączenia mogą być podobne albo zamiast wejściowych pinów analogowych mogą być zastosowane piny PWM. W przypadku zastosowania innego rodzaju przyspieszeniomierza zajrzyj do jego danych technicznych lub dokumentacji, jak go poprawnie podłączyć.

Po podłączeniu nakładki WiFi i przyspieszeniomierza możesz zacząć tańczyć rock-and-rolla.



Rysunek 8.5. Schemat podłączenia przyspieszeniomierza analogowego do Arduino

8.4.5. Szkic do komunikacji Bluetooth

Do bezprzewodowego przesyłania z Arduino danych z przyspieszeniomierza do serwera działającego na Twoim komputerze użyjesz biblioteki *WiFi*. Zaraz, serwer na Twoim komputerze? Jak to zrobić?

Zastosujesz środowisko programistyczne o nazwie **Processing**. Jeżeli nie masz go zainstalowanego na swoim komputerze, otwórz stronę *www.processing.org* i pobierz najnowszą wersję oprogramowania. Jeżeli nie korzystałeś wcześniej z tego środowiska, nie przejmuj się. Kod z listingu 8.4 wygląda znajomo i jest podobny do tego, który pisałeś już wcześniej.

Ale najpierw zajmijmy się stroną Arduino i poniższym kodem (listing 8.3).

Listing 8.3. Kod klienta dla Arduino obsługującego przyspieszeniomierz

```
#include <WiFi.h>

char ssid[] = "nazwa_sieci";      <-- Ustawienie nazwy sieci
char pass[] = "klucz_sieci";      <-- Ustawienie klucza sieci
IPAddress server_address(192,168,0,1);   <-- Ustawienie adresu serwera
int server_port = 10000;          <-- Ustawienie portu serwera
```

```
int status = WL_IDLE_STATUS;

WiFiClient client;

void setup() {
  Serial.begin(9600);
  connectToNetwork();
}

void connectToNetwork(){
  while ( status != WL_CONNECTED) {
    Serial.print("Proba polaczenia z SSID: ");
    Serial.println(ssid);

    status = WiFi.begin(ssid, pass);
    delay(10000);
  }
  printWifiStatus();
}

void loop() {
  if(status == WL_CONNECTED){
    if(!client.connected())
    {
      client.connect(server_address, server_port);
      delay(1000);
    }
    else
    {
      Serial.print("Polaczony, przesylanie danych do ");
      Serial.println(server_address);
      client.print("x: ");
      client.print(analogRead(0));
      client.print(" y: ");
      client.print(analogRead(1));
      client.print(" z: ");
      client.println(analogRead(2));
    }
    delay(20);
  }
  else{
    Serial.println("Siec WiFi niedostepna");
    connectToNetwork();
  }
}

void printWifiStatus() {
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  IPAddress ip = WiFi.localIP();
  Serial.print("Adres IP: ");
  Serial.println(ip);

  long rssi = WiFi.RSSI();
  Serial.print("Sila sygnalu (RSSI): ");
```

1. **Obiekt klienta WiFi**
2. **Zestawienie połączenia szeregowego**
3. **Próba połączenia z siecią**
4. **Połączenie z siecią**
5. **Oczekiwanie na połączenie z siecią**
6. **Sprawdzenie, czy jest połączenie z siecią**
7. **Połączenie z serwerem**
8. **Odczyt i wysłanie bieżących danych z przyspieszeniomierza**

Zwłoka 20 ms przed następną próbą połączenia

```
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Po zaimportowaniu potrzebnych bibliotek i określeniu kilku szczegółów dotyczących konfiguracji sieci i serwera tworzony jest obiekt `WiFiClient` ❶. Następnie w funkcji `setup` inicjowana jest biblioteka obsługująca transmisję szeregową na potrzeby diagnostyki ❷, po czym podejmowana jest próba nawiązania połączenia sieciowego za pomocą utworzonej funkcji `connectToNetwork` ❸. Później kod próbuje wybrać sieci WiFi za pomocą funkcji `WiFi.begin` ❹ i przez 10 sekund oczekuje na połączenie ❺. Potem następuje wejście do głównej pętli. Jeżeli udało się połączyć z siecią ❻, ale nie ma jeszcze połączenia z serwerem ❼, podejmowana jest próba połączenia się z nim. Jeżeli połączenie z serwerem już jest nawiązane, z przyspieszeniomierza odczytywane są bieżące dane i wysyłane do serwera ❽.

W listingu 8.4 przejdziemy do środowiska Processing i utworzymy serwer, który będzie przyjmował połączenia przychodzące od klientów (Arduino) i wyświetlał na ekranie odbierane komunikaty. Aby dowiedzieć się nieco więcej na temat środowiska Processing, zajrzyj do podrozdziału 13.2.

Listing 8.4. Szkic Processing odbierający od klienta Arduino dane z przyspieszeniomierza

```
import processing.net.*;                    ← Import biblioteki sieciowej Processing

int direction = 1;                          ← Ustawienie kierunku tekstu
boolean serverRunning = false;              ← Utworzenie ciągu dla danych z przyspieszeniomierza
String currentData = "";

Server myServer;                            ← Utworzenie obiektu serwera

void setup()
{
  size(400, 400);                           ← Ustawienie wielkości okna
  textFont(createFont("SansSerif", 16));
  myServer = new Server(this, 10000);       ← ❶ Inicjalizacja serwera
  serverRunning = true;
  printData();
}

void printData() {
  background(0);
  text("Dane z bezprzewodowego przyspieszeniomierza: ", 15, 25);
  text(currentData, 15, 60);
}

void draw()
{
  Client thisClient = myServer.available();     ← ❷ Sprawdzenie komunikatów od klienta
  if (thisClient != null) {
    if (thisClient.available() > 0) {
      currentData = "komunikat od: " + thisClient.ip() + " : " +
        thisClient.readString();                ← ❸ Rozpakowanie i wyświetlenie danych
      printData();
```

```
    }
  }
}
```

Po skonfigurowaniu w funkcji `setup` okna i czcionki do wyświetlania danych tworzona jest instancja serwera oczekującego na połączenia przychodzące od klientów ❶. Funkcja `draw`, podobna do funkcji `loop` Arduino, bez przerwy sprawdza, czy nadeszły połączenia od klientów ❷. Jeżeli tak, wyodrębnia komunikat (funkcja `thisClient.readString()`), dołącza do niego dodatkowe informacje, na przykład adres IP klienta, i wyświetla w oknie ❸.

8.4.6. Załadowanie i test szkicu

To wszystko, co trzeba zrobić w tym projekcie. Kliknij przycisk *Run* w środowisku Processing. Na ekranie pojawi się okno i uruchomi serwer. Załaduj szkic do Arduino. Jeżeli Twoja sieć jest skonfigurowana prawidłowo (Arduino i komputer znajdują się w tej samej sieci WiFi i w szkicu podałeś jako adres serwera adres IP swojego komputera), w oknie programu Processing powinny zacząć pojawiać się wartości odbierane z przyspieszeniomierza.

Jeżeli masz ochotę na więcej, dobrym początkiem może być wyodrębnienie z odebranego ciągu znaków poszczególnych danych z przyspieszeniomierza i przedstawienie ich w trójwymiarowym (x, y, z) widoku na ekranie.

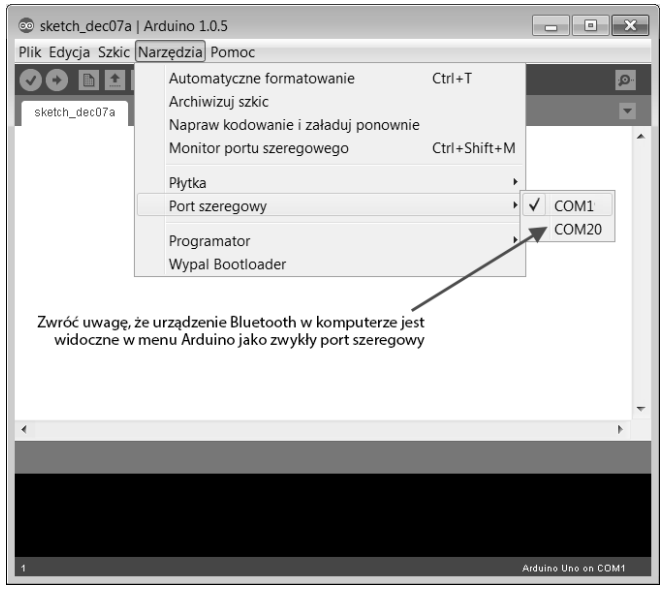
8.5. Bezprzewodowa łączność Bluetooth

Łączność WiFi nie jest jedyną dostępną formą bezprzewodowej komunikacji komputera i Arduino. Inną dobrą opcją jest Bluetooth, bezprzewodowa technologia popularnie stosowana w słuchawkach do telefonów komórkowych i urządzeniach peryferyjnych.

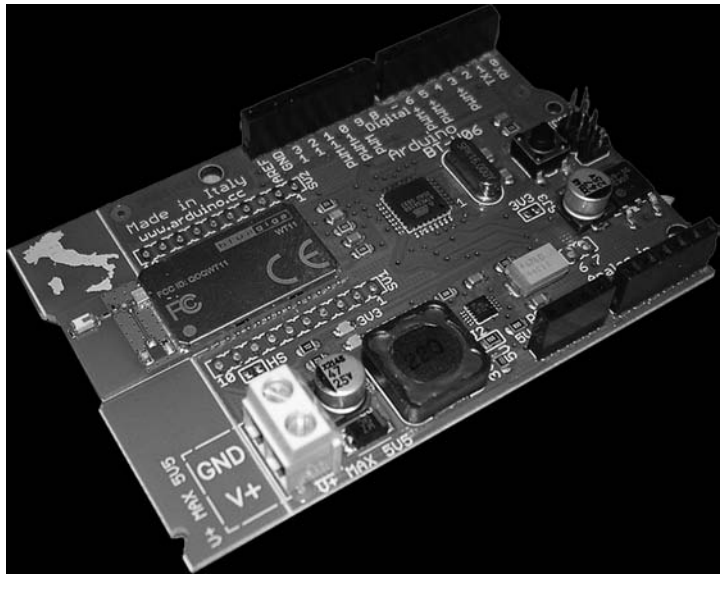
Technologia Bluetooth została opracowana przez firmę Ericsson jako otwarty standard, będący alternatywą do tradycyjnej przewodowej komunikacji szeregowej. Z założenia służy do łączności na niewielkich odległościach, tj. jej zasięg i prędkość transmisji są ograniczone. Niemniej jednak w większości przypadków zapewnia wystarczającą prędkość, jakość i bezpieczeństwo transmisji. Ponieważ urządzenia Bluetooth są widoczne w systemie operacyjnym jako wirtualne porty szeregowe (na rysunku 8.6 wewnętrzne urządzenie Bluetooth w laptopie z systemem Windows jest widoczne na liście urządzeń środowiska Arduino IDE jako zwykły port szeregowy), przesyłanie danych z Arduino jest bardzo proste. Stosowane są te same polecenia jak w zwykłej komunikacji szeregowej. Oznacza to, że niezależnie od tego, czy Twój szkic wysyła dane do komputera przez standardowe łącze USB, czy Bluetooth, kod pozostaje ten sam. My uwielbiamy taką przenośność kodu.

8.5.1. Płyta ArduinoBT

Płyta ArduinoBT (przedstawiona na rysunku 8.7) jest to Arduino z wbudowanym modułem Bluetooth do komunikacji bezprzewodowej. Płyta może pracować przy minimalnym napięciu zasilającym 1,2 V, dzięki czemu oszczędzana jest bateria. W odróżnieniu



Rysunek 8.6. Widok przedstawiający urządzenie Bluetooth komputera jako port szeregowy



Rysunek 8.7. ArduinoBT, czyli płyta Arduino z wbudowanym modułem Bluetooth

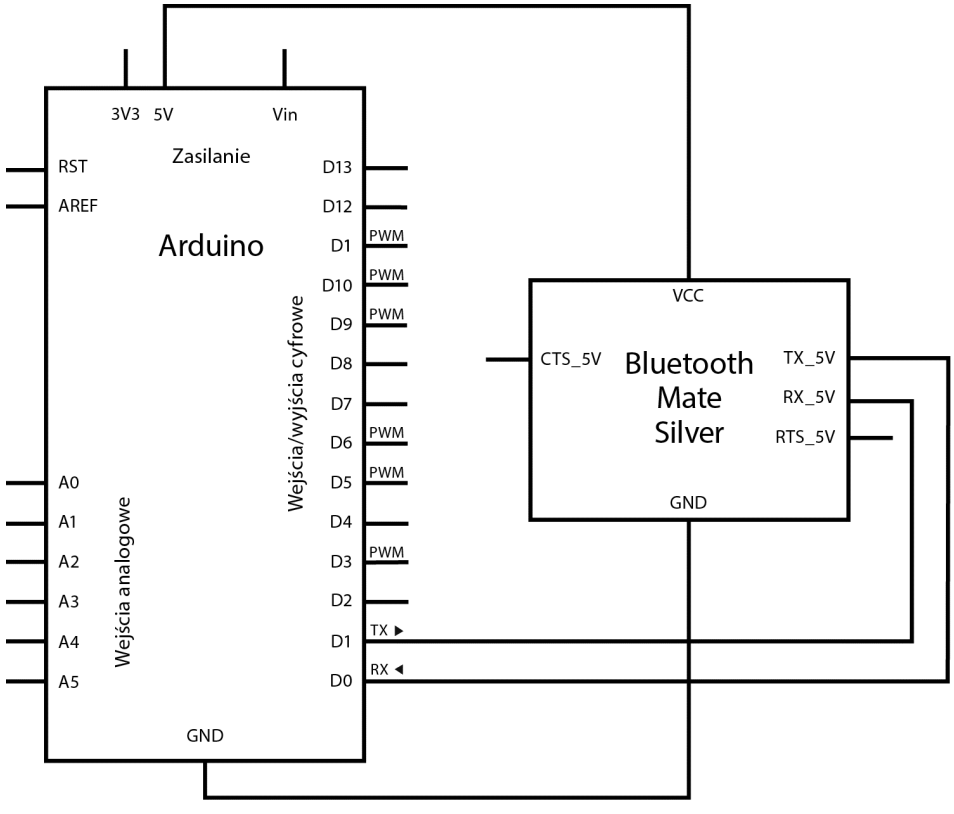
od innych modeli z podobnymi układami ATmega, płyta ArduinoBT posiada dwa dodatkowe wejścia analogowe (razem 8), ale nie są do nich przylutowane łączówki. Aby wykorzystać dodatkowe wejścia, należy bezpośrednio do ich wyprowadzeń przylutować przewody lub dwie dodatkowe łączówki (zalecany sposób).

UWAGA: Nie wolno używać pinu nr 7, ponieważ służy wyłącznie do resetowania modułu.

8.5.2. Dodawanie modułu Bluetooth

Jeżeli nie posiadasz płyty ArduinoBT lub w projekcie musi być użyty inny model płyty Arduino, to nie jesteś jedynym, który znalazł się w takiej sytuacji. Dostępnych jest wiele modułów Bluetooth, na przykład z serii BlueSMiRF (zalecanej), oferowanej przez SparkFun.

Moduły z tej serii dołącza się do linii szeregowych RX/TX Arduino, a więc w bardzo prosty sposób. Jak przedstawia rysunek 8.8, wystarczy po prostu połączyć pin Vcc modułu z pinem 5 V Arduino, a następnie piny GND obu układów. Na koniec należy połączyć linie komunikacyjne, tj. pin RX modułu BlueSMiRF z pinem TX płyty Arduino oraz TX modułu z RX płyty.



Rysunek 8.8. Schemat połączenia modułu BlueSMiRF Silver z Arduino

UWAGA: Ponieważ moduł Bluetooth przejmuje sterowanie liniami RX/TX Arduino, należy go odłączyć przed załadowaniem szkicu do płyty przez łącze USB.

8.5.3. Nawiązywanie połączenia Bluetooth

Pierwszym krokiem w uruchamianiu komunikacji Bluetooth jest skojarzenie Arduino z komputerem. Na tym etapie komputer tworzy wirtualny port szeregowy do komunikacji z Arduino za pomocą Bluetooth. Wiele dzisiejszych laptopów jest wyposażonych

we wbudowany moduł Bluetooth, ale jeżeli Twój komputer go nie posiada, możesz zakupić moduł Bluetooth z łączem USB za niecałe 150 zł — jest to znakomity prezent dla każdego hobbisty i miłośnika Arduino.

Proces kojarzenia płyty ArduinoBT z komputerem jest różny i zależy od systemu operacyjnego, ale ustawienia są takie same. Jeżeli używasz ArduinoBT, musisz odszukać w systemie urządzenie o nazwie *ARDUINOBT* — lub inne o podobnej nazwie, jeżeli posiadasz zewnętrzny moduł Bluetooth — a następnie połączyć się z nim, używając kodu *1234*. To wszystko.

Powinien pojawić się komunikat o nawiązaniu połączenia Bluetooth pomiędzy komputerem a Arduino. Teraz możesz korzystać ze środowiska Arduino IDE w zwykły sposób.

8.5.4. Szkic do komunikacji Bluetooth

Kod dla Arduino z funkcjonalnością Bluetooth niczym się nie różni od kodu wykorzystującego łącze USB. Pamiętaj, że moduł Bluetooth jest widoczny w systemie operacyjnym jako wirtualny port szeregowy, więc możesz używać dokładnie takich samych znanych Ci poleceń `Serial.print`.

Jedynym szczegółem, o którym musisz pamiętać, jest konieczność ustawienia takiej samej prędkości transmisji pomiędzy Arduino a modułem Bluetooth komputera. W przypadku płyty ArduinoBT jest to 115 200 bodów. W funkcji `setup` musisz ustawić odpowiednią prędkość przy otwieraniu połączenia szeregowego, na przykład poleceniem `Serial.begin(115200)`.

Kod przedstawiony na listingu 8.5 nie robi niczego wyjątkowego, ale pokazuje, jak prosty może być szkic wykorzystujący komunikację Bluetooth. Konfigurowana jest odpowiednia prędkość transmisji, po czym co sekundę jest rozsyłany przez bezprzewodowy port szeregowy komunikat zawierający losową liczbę. Ciekawą cechą kodu jest możliwość użycia go w Twoich obecnych projektach. Wystarczy tylko zmienić polecenia `print` i można wysyłać odczyty z dowolnego czujnika zastosowanego w projekcie.

Listing 8.5. Szkic testowy Bluetooth

```
void setup(){
  Serial.begin(115200);
}

void loop(){
  Serial.print(random(1024));
  delay(1000);
}
```

> **UWAGA:** Jeżeli komunikaty nie są wysyłane poprawnie, sprawdź prędkość transmisji. Niektóre moduły Bluetooth mogą stosować prędkość 9600 bodów lub inną.

Gratulacje, poznałeś już większość sposobów komunikacji przewodowej i bezprzewodowej dostępnych w Arduino. Komunikacja jest jednak szerokim pojęciem, a przesyłanie danych przez sieć nie jest jej jedyną możliwą formą. W kolejnych częściach

rozdziału poznasz ciekawy kanał komunikacyjny o nazwie SPI (ang. *Serial Peripheral Interface*, szeregowy interfejs urządzeń peryferyjnych). Jest to bardzo skuteczna metoda komunikacji Arduino z innymi urządzeniami.

8.6. Interfejs SPI

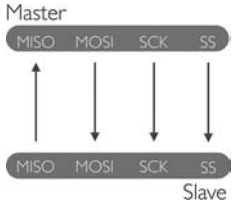
Interfejs SPI jest to protokół synchronicznej transmisji danych, umożliwiający szybką komunikację na niewielkich odległościach pomiędzy kilkoma mikrokontrolerami i urządzeniami peryferyjnymi. Zastosowana jest w nim analogia do pana i niewolnika (*master – slave*), gdzie jedno z urządzeń (zazwyczaj mikrokontroler) jest typu master i steruje dołączonymi urządzeniami peryferyjnymi typu slave.

Zazwyczaj komunikacja SPI jest realizowana za pomocą trzech linii:

- MISO — wejście urządzenia master, wyjście urządzenia slave;
- MOSI — wyjście urządzenia master, wejście urządzenia slave;
- SCK — linia zegara.

Działanie tych trzech linii jest dość proste: linia MISO służy do przesyłania danych z urządzenia slave do urządzenia master, linia MOSI z urządzenia master do urządzenia slave, a SCK (oznaczana niekiedy SCLK lub Clock) jest sygnałem zegara synchronizującym transmisję danych.

Każde urządzenie peryferyjne posiada również pin SS (ang. *slave select*, wybór urządzenia slave), który może być użyty do wybrania urządzenia do komunikacji. Pojawienie się poziomu niskiego (`LOW`) na pinie SS informuje urządzenie slave, że nastąpi komunikacja z urządzeniem master, natomiast stan wysoki (`HIGH`) oznacza, że należy zignorować komunikację. Dzięki pinowi SS można podłączyć kilka urządzeń wykorzystujących te same trzy linie komunikacyjne i przełączać się na jedno z nich w danej chwili poprzez ustawienie odpowiednio stanu wysokiego lub niskiego.



Rysunek 8.9. Kanał komunikacyjny SPI

Rysunek 8.9 zawiera obrazowe przedstawienie komunikacji SPI.

Przydatną cechą komunikacji SPI jest niezależność linii MISO oraz MOSI, dzięki czemu można ich używać jednocześnie (w odróżnieniu od innych protokołów, na przykład I^2C, wykorzystujących jedną linię do komunikacji w obu kierunkach).

Przy konfigurowaniu połączenia SPI w Arduino skorzystaj z tabeli 8.6.

8.6.1. Biblioteka SPI

Aby ułatwić Ci pracę, przyjaciele z Arduino stworzyli bibliotekę SPI, dzięki której możesz szybko skonfigurować komunikację. Tabela 8.7 zawiera listę funkcji z tej biblioteki, z których będziesz korzystać podczas komunikacji z urządzeniami za pomocą protokołu SPI.

Tabela 8.6. Opis linii SPI płyty Arduino

Linia SPI	Pin Arduino (oprócz modelu Mega)	Pin Arduino Mega
Slave select (wybór urządzenia slave) lub *chip select* (wybór układu) (SS/CS)	10	53
Wyjście urządzenia master, wejście urządzenia slave lub wejście szeregowe (MOSI/SDI)	11	51
Wejście urządzenia master, wyjście urządzenia slave lub wyjście szeregowe (MISO/SDO)	12	50
Sygnał zegara (SCK/SCLK)	13	52

Tabela 8.7. Funkcje biblioteki SPI

Funkcja	Opis
`begin()`	Inicjuje szynę SPI, ustawia piny SCK, MOSI oraz SS jako wyjścia (stan niski na SCK i MOSI, wysoki na SS).
`end()`	Blokuje szynę SPI.
`setBitOrder(`*kolejność*`)`	Ustawia kolejność przesuwania bitów podczas wysyłania i odbierania danych przez szynę SPI. Dopuszczalne parametry to `LSBFIRST` (*least-significant bit first*, najpierw najmniej znaczący bit) i `MSBFIRST` (*most-significant bit first*, najpierw najbardziej znaczący bit).
`setClockDivider(amt)`	Ustawia dzielnik zegara SPI w odniesieniu do zegara systemowego. Domyślną wartością jest `4` (jedna czwarta częstotliwości zegara systemowego). Inne dopuszczalne wartości to `2`, `4`, `8`, `16`, `32`, `64` i `128`.
`byte transfer(`*dane*`)`	Przesyła przez szynę SPI jeden bajt danych w obu kierunkach. Parametr *dane* oznacza bajt do wysłania. Funkcja zwraca bajt odczytany z szyny.
`setDataMode(`*tryb*`)`	Ustawia tryb zegara (biegunowość i fazę). Dopuszczalne wartości to `SPI_MODE0`, `SPI_MODE1`, `SPI_MODE2`, `SPI_MODE3`.

Po zapoznaniu się z powyższymi funkcjami możesz przystąpić do podłączania urządzenia peryferyjnego do Arduino i nawiązać z nim komunikację przez interfejs SPI.

8.6.2. Urządzenia SPI i potencjometry cyfrowe

Za pomocą interfejsu SPI mogą komunikować się urządzenia różnego typu, począwszy od kolorowych wyświetlaczy LCD, poprzez magnetometry, na sterowanych kolorowych panelach LED skończywszy. Opisana wcześniej nakładka Ethernet i czytnik kart SD również wykorzystują interfejs SPI do komunikacji z Arduino. Podstawowa zasada komunikacji w przypadku wszystkich wyżej wymienionych urządzeń jest taka sama. Dlatego w celu zademonstrowania zastosowania interfejsu SPI użyjemy potencjometru cyfrowego AD5206 firmy Analog Devices.

Potencjometry cyfrowe są bardzo podobne do swoich analogowych odpowiedników, których używamy na co dzień, z tym jednym wyjątkiem, że są regulowane elektronicznie, a nie ręcznie. Model AD5206 zawiera sześć wbudowanych niezależnych potencjometrów, z których można jedocześnie korzystać. Na rynku dostępnych jest również wiele potencjometrów w innych konfiguracjach.

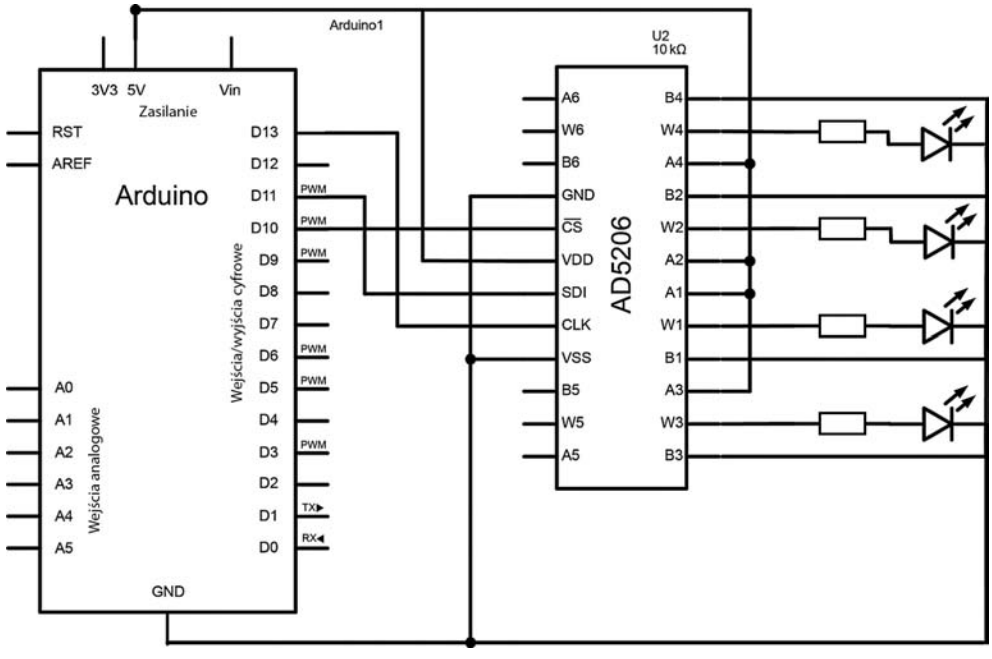
W tym przykładzie wykorzystamy cztery wbudowane potencjometry do sterowania jasnością czterech niezależnych diod LED. Jak zapewne się domyślasz, potencjometr cyfrowy może być użyty do sterowania dowolnym urządzeniem wymagającym zmieniającego się napięcia.

Przejdźmy zatem do rzeczy i zobaczmy, jak połączyć ze sobą wszystkie elementy.

8.6.3. Schemat układu i połączenia elementów

Pierwszą rzeczą, którą musisz zrobić, jest sprawdzenie danych technicznych zastosowanego potencjometru cyfrowego. Komunikacja SPI wymaga czterech pinów (MOSI/SDI, MISO/SDO, SS/CS oraz SCK), musisz więc je odnaleźć na swoim potencjometrze, korzystając z dokumentacji.

Schemat na rysunku 8.10 przedstawia układ pinów potencjometru cyfrowego AD5206 oraz sposób jego połączenia z Arduino i diodami LED.



Rysunek 8.10. Cztery diody LED sterowane za pomocą cyfrowego potencjometru AD5206

Najpierw musisz wykonać połączenia zasilające. Połącz pin VDD potencjometru AD5206 z pinem 5 V Arduino, jak również połącz ze sobą piny GND. Następnie połącz pin VSS potencjometru AD5206 z pinem GND Arduino.

Po wykonaniu głównych połączeń zasilających możesz zająć się połączeniami SPI. Najpierw połącz pin CS potencjometru (ang. *chip/slave select*) z pinem cyfrowym nr 10 płyty Arduino, a następnie pin SDI (MOSI) potencjometru z pinem cyfrowym nr 11 Arduino. Ponieważ będziesz korzystać tylko z linii MOSI, nie musisz podłączać pinu MISO (SDO).

Pozostał tylko pin SCK, wymagany w komunikacji SPI, zatem podłącz go do pinu cyfrowego nr 13 płyty Arduino.

To jest wszystko, co dotyczy zasilania układów i komunikacji SPI. Trzeba jeszcze wykonać połączenia potencjometrów. Na rysunku 8.10 każdy z sześciu potencjometrów posiada trzy piny, na przykład A1, B1 oraz W1. Możesz je traktować jak końcówki zwykłego potencjometru. Wszystkie piny A podłącz do pinu 5 V płyty Arduino, wszystkie piny B do pinu GND. Piny W to suwaki, na których pojawia się zmieniające się napięcie. Piny te podłącz do dodatnich końcówek (anod) diod LED poprzez rezystor (wartość 220 Ω powinna być odpowiednia, ale możesz ją obliczyć, aby uzyskać odpowiednią jasność konkretnych zastosowanych diod LED). Podłącz ujemne końcówki (katody) diod LED do pinu GND Arduino i gotowe!

Teraz przyjrzyjmy się prostemu szkicowi zmniejszającemu i zwiększającemu jasność diod LED i zobaczmy potencjometry cyfrowe w akcji.

8.6.4. Szkic cyfrowego sterownika diod LED

Po połączeniu wszystkich elementów czas na uruchomienie potencjometrów cyfrowych. W poniższym szkicu kolejno regulowana jest jasność każdej diody. W tym celu każda z nich jest niezależnie wybierana za pomocą interfejsu SPI.

Skopiuj dokładnie listing 8.6 do środowiska Arduino IDE.

Listing 8.6. Potencjometry cyfrowe do regulacji jasności diod LED za pomocą interfejsu SPI

```
#include <SPI.h>                    <-- Import biblioteki SPI
const int slaveSelectPin = 10;      <-- Określenie pinu slave select
int numLeds = 4;

void setup()
{
  pinMode(slaveSelectPin, OUTPUT);  <-- 1 Ustawienie pinu slave select jako wyjścia
  SPI.begin();                      <-- 2 Włączenie komunikacji SPI
}

void setLed(int reg, int level)
{
  digitalWrite(slaveSelectPin, LOW);    <-- 3 Ustawienie stanu niskiego na pinie slave select
  SPI.transfer(reg);                    <-- 4 Wybranie rejestru
  SPI.transfer(level);                  <-- 5 Wysłanie danych
  digitalWrite(slaveSelectPin, HIGH);   <-- 6 Ustawienie stanu wysokiego na pinie slave select

}

void loop()
{
  for(int i=0; i<numLeds; i++)
  {
    for (int j=50; j<=255; j++)
    {
      setLed(i,j);
      delay(20);
    }
```

```
    delay(500);
    for (int j=255; j>=50; j--)
    {
      setLed(i,j);
      delay(20);
    }
  }
}
```

Po zaimportowaniu biblioteki *SPI* i zadeklarowaniu zmiennych konfigurowany jest jako wyjście pin wyboru urządzenia slave ❶. Następnie za pomocą funkcji `begin` nawiązywana jest komunikacja SPI ❶.

Sterowanie diodą LED za pomocą interfejsu SPI jest procesem złożonym z czterech kroków, dlatego w tym celu utworzona została funkcja `setLed()`, która wykonuje wszystkie kroki przy każdym wywołaniu:

1. Na pinie SS jest ustawiany stan niski (`LOW`) w celu wybrania urządzenia slave, z którym będzie nawiązana komunikacja ❸. Następnie interfejs SPI oczekuje dwóch komunikatów: jednego z informacją o rejestrze, który zostanie użyty, a drugi z wpisywaną do rejestru wartością.
2. Potencjometr AD5206 posiada sześć rejestrów, a diody LED są dołączone do rejestrów o numerach 0 – 3. Można więc wywołać funkcję `SPI.transfer()` z parametrem oznaczającym numer diody LED (0 – 3), która ma być wysterowana ❹.
3. Funkcja `SPI.transfer()` jest wywoływana ponownie z liczbą (0 – 255) określającą jasność diody ❺. Układ AD5206 zawiera cyfrowe potencjometry 8-bitowe, dlatego maksymalna wartość jasności jest równa 255.
4. Po przesłaniu danych można na wyjściu SS ustawić stan wysoki (`HIGH`) i to wszystko ❻.

W funkcji `loop()` wybierana jest kolejno każda dioda i co 20 milisekund jej jasność jest zwiększana, a następnie co 20 milisekund zmniejszana. Możesz dowolnie zmieniać kod i uzyskiwać przy użyciu diod LED inne ciekawe efekty wizualne.

To wszystko na temat kodu. Załaduj szkic do Arduino i diody powinny zmieniać jasność wraz ze zmieniającym się napięciem bez ręcznego sterowania i bez użycia sygnału PWM.

Gratulacje! Poznałeś właśnie jedno z bardziej zaawansowanych zastosowań Arduino. Teraz przyjrzymy się innemu ważnemu obszarowi, często wykorzystującemu komunikację *SPI*: logowaniu danych.

8.7. Rejestrowanie danych

Czujniki doskonale nadają się do zbierania danych o stanie jakiejś rzeczy. Ale jak zmieniają się wskazania czujnika w czasie i czy ta informacja jest przydatna?

Zastanówmy się na chwilę nad muzyką. Muzyka to zjawisko fizyczne, które można traktować jako ciąg zdarzeń pojawiających się w określonym przedziale czasu. Każde

zdarzenie może mieć różne cechy, takie jak ton (określający, jak wysoka lub niska jest nuta), amplitudę (głośność) i barwę (jakość), opisujące brzmienie każdej nuty lub dźwięku.

Każda nuta lub zdarzenie z osobna nie stanowi muzyki. Dopiero gdy są kolejno odgrywane, zgodnie z kompozycją, tworzone są pomiędzy nimi różne relacje. Powstaje melodia i rytm, w przestrzeni tworzone są struktury wyższego rzędu. Nawet amator z „dębowym" uchem z łatwością rozpozna te relacje. Chociaż nie możemy ich opisać przy użyciu terminów muzycznych, potrafimy je odczuć i wyrazić. Jest tak dlatego, że ludzki mózg posiada pamięć długo- i krótkoterminową, mogącą przechowywać relacje między odgrywanymi dźwiękami, a nawet porównywać je z relacjami w bibliotece dźwięków usłyszanych w przeszłości. Nasze mózgi mogą przypomnieć sobie i połączyć zdarzenia!

Podobnie jeżeli chcesz poznać strukturę i relacje wyższego rzędu pomiędzy danymi odbieranymi z czujników, musisz je zbierać przez pewien okres czasu, a potem przeanalizować. W tej części rozdziału przyjrzymy się, jak rejestrować dane za pomocą Arduino.

8.7.1. Rodzaje pamięci

Aby rejestrować dane lokalnie na Arduino, musisz mieć gdzie je zapisać. Zazwyczaj wykorzystuje się w tym celu jakąś zewnętrzną pamięć, na przykład w Twoim komputerze, telefonie komórkowym lub pendrivie. Pamięć zewnętrzna ma tę cenną cechę, że dane w niej zapisane pozostają zapamiętane nawet po wyłączeniu zasilania. Ta zasada dotyczy również wbudowanej w Arduino pamięci EEPROM.

Zawartość pamięci EEPROM jest przechowywana nawet po wyłączeniu zasilania Arduino, ale jej wielkość jest bardzo ograniczona. W zależności od modelu płyty dostępnych jest od 512 do 4096 bajtów. Dlatego pamięć EEPROM nadaje się do przechowywania zmiennych i niewielkich ilości danych, które muszą być dostępne pomiędzy kolejnymi włączeniami Arduino, ale nie nadaje się do większości zastosowań obejmujących rejestrowanie danych. Dlatego często będziesz korzystać z zewnętrznych rozwiązań, takich jak karta SD (ang. *Secure Digital*, bezpieczny cyfrowy zapis) lub pamięć USB.

8.7.2. Karty SD i biblioteka SD

Użycie karty SD jest jednym z najlepszych sposobów przechowywania danych w pamięci zewnętrznej. Dostępne są karty o dużych pojemnościach, oferujących mnóstwo przestrzeni nie tylko na rejestrowane dane, ale również na innego rodzaju pliki (na przykład muzykę lub obrazy). Dla płyty Arduino jest dostępna biblioteka *SD*, dzięki której odczytywanie i zapis danych na karcie SD są proste.

Tabela 8.8 zawiera przegląd najważniejszych funkcji klasy `SD` z biblioteki *SD*, ale dostępnych jest znacznie więcej funkcji klasy `File`, służących do odczytu i zapisu plików. Tabela 8.9 zawiera listę najważniejszych funkcji tej klasy, natomiast pełna lista jest dostępna w dokumentacji w Internecie pod adresem *www.arduino.cc/en/Reference/SD*.

Niezależnie od tego, czy używasz nakładki SD, czy czytnika wbudowanego w nakładkę Ethernet, biblioteki *SD* używa się w taki sam sposób. Jeżeli dobrze znasz funkcje opisane wcześniej, to jesteś w pełni gotowy do napisania szkicu rejestrującego rzeczywiste dane.

Tabela 8.8. Funkcje klasy SD z biblioteki SD

Funkcja	Opis
`begin(chipSelect)`	Inicjuje bibliotekę *SD* i kartę, opcjonalnie można wskazać pin do wyboru układu.
`exists()`	Sprawdza, czy plik lub katalog istnieje na karcie SD.
`mkdir("/katalog/do/utworzenia")` `rmdir("/katalog/do/usuniecia")`	Tworzy lub usuwa katalog z karty SD.
`open("plik/do/otwarcia", tryb)` `remove("plik/do/usuniecia")`	Otwiera lub usuwa plik o zadanej ścieżce. Przy otwieraniu pliku można określić tryb `FILE_READ` (odczyt) lub `FILE_WRITE` (zapis), jeżeli zachodzi potrzeba ograniczenia do niego dostępu tylko do odczytu lub tylko do zapisu.

Tabela 8.9. Funkcje klasy File z biblioteki SD

Funkcja	Opis
`available()`	Sprawdza, czy w pliku dostępne są bajty do odczytu.
`close()`	Zamyka plik i sprawdza, czy wszystkie dane zostały zapisane na karcie SD.
`flush()`	Zapisuje fizycznie dane na karcie SD. Funkcja wykorzystywana przez `close()`.

`print()`
`println()`
`write()`
	Zapisuje dane do pliku.

`read()`	Odczytuje bajt z pliku.

8.7.3. Szkic rejestrujący na karcie SD dane z czujnika

W tej części rozdziału utworzysz prosty szkic rejestrujący w pliku na karcie SD dane z czujnika dołączonego do jednego z wejść analogowych (nr 0). Listing 8.7 przedstawia taki przykładowy szkic.

Listing 8.7. Rejestrator danych na karcie SD

```
#include <SD.h>                    ← Dołączenie biblioteki SD

const int chipSelect = 4;

void setup()
{
  Serial.begin(9600);              ← Zainicjowanie portu szeregowego do diagnostyki
  Serial.print("Inicjalizacja karty SD...");
  pinMode(10, OUTPUT);             ← Ustawienie domyślnego pinu slave select jako wyjścia

  if (!SD.begin(chipSelect)) {     ← (1) Sprawdzenie karty SD
    Serial.println("Blad karty lub karta niedostepna");
    return;
  }
  Serial.println("Karta zainicjowana.");
}

void loop()
{
```

```
  String dataString = "";

  int data = analogRead(0);
  dataString += String(sensor);

  File dataFile = SD.open("datalog.txt", FILE_WRITE);

  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
    Serial.println(dataString);
  }
  else {
    Serial.println("Blad otwarcia pliku datalog.txt");
  }
}
```

❷ Utworzenie zmiennej String na dane z czujnika

❸ Odczyt danych z czujnika i zapisanie w dataString

❹ Otwarcie pliku, zapis danych i zamknięcie

❺ Wyświetlenie danych na monitorze w celu diagnostyki

Rejestrowanie danych nie może już być prostsze. Najpierw musisz sprawdzić, czy karta SD jest dostępna ❶. Jeżeli tak, wszystko jest gotowe do działania, a jeżeli karta nie jest dostępna, następuje powrót z programu.

Po skonfigurowaniu karty można zacząć wykonywać główną pętlę, utworzyć obiekt typu `String` do przechowywania danych ❷, a następnie odczytać bieżące dane z czujnika i umieścić je w tym obiekcie ❸. Dalej ma miejsce otwarcie pliku, zapisanie w nim danych i zamknięcie ❹. Na potrzeby diagnostyki błędów możesz dane wyświetlić na monitorze portu szeregowego ❺.

Do wejścia analogowego możesz dołączyć dowolny czujnik, na przykład potencjometr, czujnik temperatury lub dalmierz ultradźwiękowy. Gdy dane zostaną zapisane, umieść kartę SD w czytniku swojego komputera i otwórz plik w dowolnym edytorze tekstowym lub środowisku programistycznym. Teraz możesz przedstawić dane w graficznej formie, dzięki której z pewnością lepiej poznasz ich przebieg.

Jeżeli chcesz udostępnić zarejestrowane dane na zewnątrz, to przekazywanie karty SD nie jest najlepszym sposobem. Do tego celu możesz użyć serwisu Xively, usługi internetowej do udostępniania bieżących danych użytkownikom na całym świecie.

8.8. Serwis Xively

Xively, dawniej Pachube, jest bardzo interesującym otwartym serwisem służącym do udostępniania w formie kanałów bieżących danych w Internecie. Możesz w nim tworzyć kanały publiczne i prywatne, do których mogą dołączać się serwery i klienci, aby wysyłać i odczytywać dane z dowolnego miejsca na świecie. Dane są przesyłane w formacie czytelnym dla człowieka (XML), a serwis Xively oferuje interesujące metody ich wizualizacji i monitorowania przez Internet.

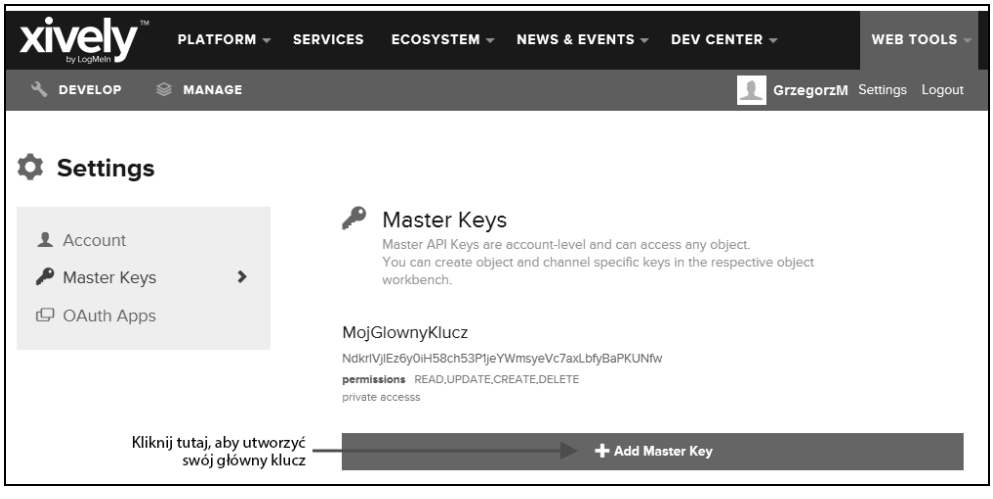
W tym podrozdziale zastosujesz płytę Arduino wyposażoną w łącze Ethernet do wysyłania danych z czujnika do kanału w serwisie Xively, jak również do odbierania danych innego kanału. A więc zaczynamy.

8.8.1. Tworzenie konta i pobieranie klucza API

Pierwszym krokiem jest utworzenie konta w serwisie Xively pod adresem *www.xively.com* i pobranie klucza API. Klucz API jest potrzebny do tworzenia, usuwania, pobierania, edycji i aktualizacji kanałów danych.

Jeżeli udostępniasz swoje projekty innym użytkownikom, powinieneś utworzyć swój prywatny główny klucz API (*Master API*) i przy użyciu panelu sterującego Xively utworzyć nowy klucz do udostępniania danych. Dzięki kluczom udostępniającym masz pełną kontrolę nad danymi i funkcjonalnościami dostępnymi dla innych użytkowników.

Po utworzeniu konta zaloguj się do serwisu i korzystając z głównego menu, przejdź do strony *Master Keys* (przedstawionej na rysunku 8.11). Możesz tam utworzyć swój główny klucz *Master API*. Po utworzeniu klucza zanotuj go i trzymaj pod ręką, gdyż będzie potrzebny później.



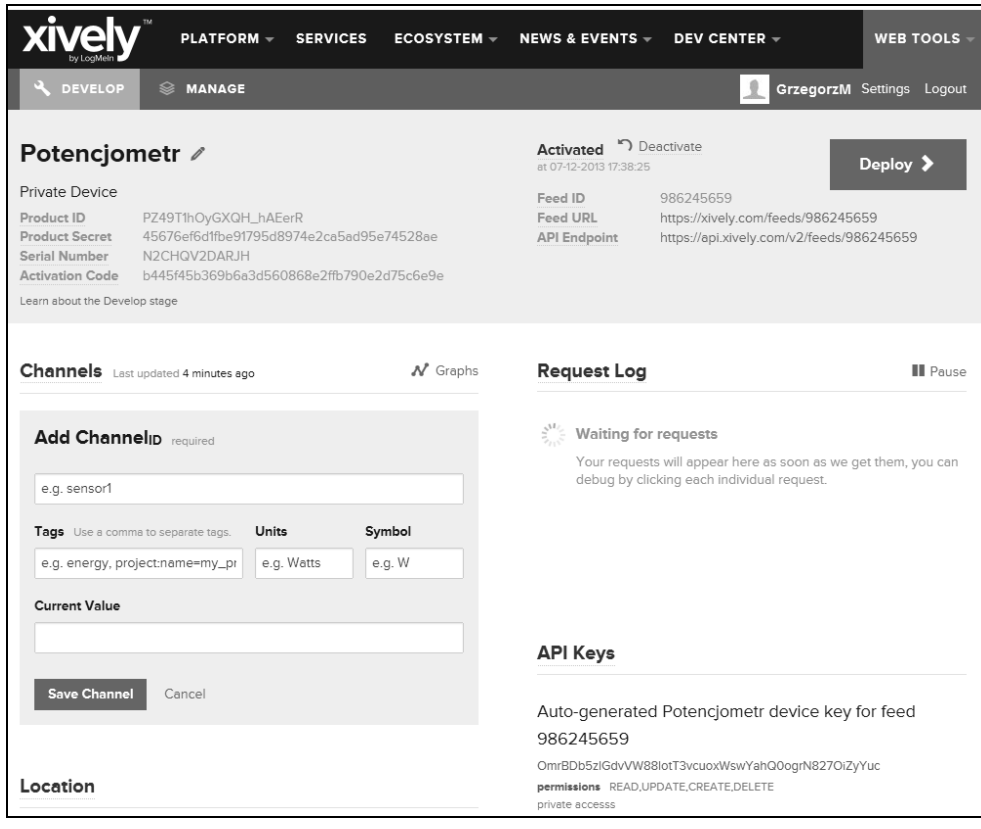
Rysunek 8.11. Widok głównego klucza w serwisie Xively

8.8.2. Tworzenie nowego kanału danych

Teraz utwórz nowy kanał, do którego będziesz wysyłać dane odczytane z czujnika.

Jeżeli jesteś zalogowany do swojego konta, wybierz polecenie menu *WEB TOOLS/DEVELOP*, a następnie kliknij dużą ikonę *Add Device*. Wprowadź nazwę i opis czujnika i kliknij przycisk *Add Device*. W następnym oknie kliknij przycisk *Add Channel*. Pojawi się widok podobny do przedstawionego na rysunku 8.12. Nadaj kanałowi tytuł i kilka tagów opisu (oczywiście możesz je później zmienić), zanotuj identyfikator kanału *Feed ID* i kliknij przycisk *Save Channel*. Twój nowy kanał został utworzony.

Twój kanał jest skonfigurowany i gotowy do odbierania informacji. Mając utworzone konto, klucz API (*API Key*) i identyfikator kanału (*Feed ID*), możesz przejść do szkicu rejestrującego dane, opisanego w punkcie 8.8.3. Jeżeli nie zapisałeś identyfikatora kanału, możesz go odnaleźć po otwarciu jego strony (po prostu kliknij w panelu Xively nazwę kanału).



Rysunek 8.12. Tworzenie nowego kanału do rejestrowania danych czujnika w serwisie Xively

8.8.3. Szkic do rejestrowania danych z czujnika w serwisie Xively

Komunikacja z serwisem Xively jest prosta. W tym przykładzie będziesz rejestrować co 10 sekund odczyty z potencjometru.

> **UWAGA:** Aby lepiej przedstawić zasadę komunikacji z serwisem Xively, w tym przykładzie nawiązanie połączenia i wysyłanie wszystkich danych jest realizowane ręcznie. Po zapoznaniu się z funkcjonowaniem komunikacji możesz oszczędzić kilka wierszy kodu. W tym celu zajrzyj do opisu doskonałej biblioteki *Xively*, dostępnej pod adresem *https://xively.com/dev/libraries*.

Wprowadź kod z listingu 8.8 do środowiska Arduino IDE.

Listing 8.8. Kod rejestrujący dane z czujnika w serwisie Xively

```
#include <SPI.h>
#include <Ethernet.h>

#define APIKEY "TWOJ KLUCZ API"
#define FEEDID 12345
#define PROJECTNAME "Arduino w akcji"
```

1 Konfiguracja kanału Xively

```
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
IPAddress ip(192,168,0,120);
EthernetClient client;

long lastConnectionTime = 0;
boolean lastConnected = false;
const int postingInterval = 10000;

void setup() {
  if (!Ethernet.begin(mac)) {
    Ethernet.begin(mac, ip);
  }
  Serial.begin(9600);
}
void loop() {
  int sensorReading = analogRead(A0);

  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  if (!client.connected() && lastConnected) {
    Serial.println();
    Serial.println("Rozlaczony.");
    client.stop();
  }

  if(!client.connected() && (millis() - lastConnectionTime >
  ↪postingInterval)) {
    sendData(sensorReading);
  }

  lastConnected = client.connected();
}

void sendData(int thisData) {
  if (client.connect("api.xively.com", 80)) {
    Serial.println("Laczenie...");
    client.print("PUT /v2/feeds/");
    client.print(FEEDID);
    client.println(".csv HTTP/1.1");
    client.println("Host: api.xively.com ");
    client.print("X-ApiKey: ");
    client.println(APIKEY);
    client.print("User-Agent: ");
    client.println(PROJECTNAME);
    client.print("Content-Length: ");

    int thisLength = 8 + getLength(thisData);
    client.println(thisLength);

    client.println("Content-Type: text/csv");
    client.println("Connection: close");
    client.println();
```

❷ **Konfiguracja ustawień sieciowych**

Utworzenie obiektu serwera WWW

Interwał aktualizacji danych w Xively

❸ **Otwarcie połączenia sieciowego**

Otwarcie połączenia szeregowego do diagnostyki

❹ **Zapamiętanie bieżących danych z czujnika**

❺ **Wyświetlenie odebranych danych w celu diagnostyki**

❻ **Zatrzymanie klienta, jeżeli nie jest połączony**

❼ **Wywołanie sendData, jeżeli przyszła pora**

Ustawienie zmiennej Boolean dla timera

❽ **Połączenie z kanałem Xively**

❾ **Polecenie PUT i inicjacja komunikacji**

❿ **Wysłanie klucza API**

```
    client.print("czujnik1, ");
    client.println(thisData);    ⓫ Wysłanie danych do Xively
  }
  else {
    Serial.println("Blad polaczenia.");
    Serial.println("Rozlaczony.");
    client.stop();
  }

  lastConnectionTime = millis();    ⓬ Zapamiętanie bieżącego czasu
}

int getLength(int someValue) {
  int digits = 1;
  int dividend = someValue /10;
  while (dividend > 0) {
    dividend = dividend /10;
    digits++;
  }
  return digits;
}
```

Najpierw musisz zmienić ustawienia swojego kanału w serwisie Xively ❶. Upewnij się, że Twój klucz API jest zamknięty w cudzysłowie, ponieważ będzie przekazywany jako ciąg typu `String`. Następnie skonfiguruj ustawienia sieciowe, podaj adres MAC i ręcznie ustaw adres IP ❷, jeżeli podczas nawiązywania połączenia nie jest dostępna usługa DHCP ❸.

Dalej następuje wejście w główną funkcję `loop()`, w której odczytywane są bieżące dane z czujnika ❹. Funkcja sprawdza, czy nadeszły dane, i wyświetla je na konsoli na potrzeby diagnostyki kodu ❺. Jeżeli byłeś wcześniej połączony z serwisem, ale już się rozłączyłeś, musisz zatrzymać klienta ❻. Na koniec jeżeli nie jesteś połączony, a upłynął pewien odstęp czasu pomiędzy wysyłanymi komunikatami, musisz się ponownie połączyć z serwisem i wysłać bieżące dane ❼.

W kodzie wywoływana jest funkcja `sendData()`. Jest to funkcja pomocnicza nawiązująca połączenie z kanałem Xively ❽ i wysyłająca dane. Najpierw musisz wysłać polecenie `PUT` inicjujące połączenie z kanałem ❾, a następnie podać swój klucz API ❿. Dalej funkcja formatuje komunikat (dane) i przesyła je do serwisu Xively ⓫. Na koniec zapisywany jest czas wysłania komunikatu ⓬, na podstawie którego obliczany jest czas wysłania następnego komunikatu.

8.8.4. Załadowanie i test szkicu

Teraz możesz podłączyć do płyty nakładkę Ethernet i przewód USB i załadować szkic. To cała konfiguracja. Otwórz widok swojego kanału na stronie Xively. Powinieneś zobaczyć zapisywane w nim dane z czujnika. Jeżeli nie widzisz danych, otwórz środowisko Arduino IDE i kliknij ikonę monitora portu szeregowego, aby zdiagnozować problem.

8.9. Podsumowanie

W tym rozdziale zacząłeś poznawać szczegóły szerokiego spektrum form komunikacji dostępnych w Arduino. Zacząłeś od podłączenia płyty do sieci i do Internetu. Za pomocą nakładki Ethernet zamieniłeś Arduino w serwer dostarczający użytkownikowi informacji na każde jego żądanie. Potem wykonałeś następny krok i połączyłeś Arduino z portalem Twitter, w którym rozsyłałeś na cały świat komunikaty o naciśnięciach przycisku. Ponieważ bezprzewodowa komunikacja z Arduino przydaje się w bardzo wielu zastosowaniach, dowiedziałeś się również, jak wykorzystać do tego celu technologie WiFi i Bluetooth.

Kanały komunikacyjne dostępne w Arduino są dalekosiężne i dostępnych jest wiele sposobów przesyłania danych pomiędzy innymi urządzeniami i akcesoriami peryferyjnymi. Jednym z takich kanałów jest interfejs SPI i w tym rozdziale dowiedziałeś się, jak inne urządzenia i nakładki (na przykład karta SD lub nakładka Ethernet) wykorzystują ten interfejs do komunikacji z Arduino. Wykorzystałeś nawet SPI do sterowania cyfrowymi potencjometrami.

Mając do dyspozycji wszystkie wszechstronne metody komunikacji, przyjrzeliśmy się następnie rejestrowaniu danych, tj. kluczowemu aspektowi komunikacji i interakcji między urządzeniami. Dowiedziałeś się nie tylko, jak przechowywać informację w lokalnej pamięci Arduino, ale również jak ją rejestrować w Internecie, korzystając z uniwersalnego serwisu Xively.

Teraz tylko od Ciebie zależy, jakie dane chcesz udostępnić i w jaki sposób. Możesz obsługiwać zdalnie czujniki w różnych otoczeniach, a ich odczyty przesyłać do kanału i później przedstawiać w swojej galerii. Możesz podłączyć się do innych publicznych kanałów rejestrujących różnorakie dane, począwszy od wstrząsów sejsmicznych, poprzez temperaturę, wilgotność, punkt rosy, na wilgotności gleby w ogrodzie skończywszy, a następnie wykorzystywać je w swojej wirtualnej przestrzeni. Każdy taki scenariusz oferuje niemal nieograniczone możliwości przesyłania i interpretacji wszelkiego typu danych.

Projekty opisane w tym rozdziale stanowią zaledwie wierzchołek góry lodowej. Opisane formy komunikacji możesz wykorzystywać na wiele sposobów, a ich nieprzebrana liczba sprawia, że Arduino jest potężnym narzędziem komunikacyjnym. Zastanów się nad celem swojego projektu i zadecyduj, czy podłączenie go do Internetu i udostępnienie całemu światu do wglądu i wykorzystania go będzie pożyteczne. Czy możesz w projekcie wykorzystać dostępne zasoby? Czy jest potrzebna komunikacja z innymi urządzeniami peryferyjnymi? Czy przyda się rejestrowanie jakichś danych zmieniających się w czasie? Arduino daje Ci do ręki wszelkie niezbędne narzędzia.

Ale nie zatrzymujmy się w tym miejscu. Teraz, w rozdziale 9., zobaczysz, jak Arduino może współpracować z urządzeniami do gier, takimi jak kontrolery Nintendo Wii Nunchuk oraz Xbox.

Skorowidz

F

G

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

ARDUINO W AKCJI

Arduino to niesamowita platforma, która otworzyła świat elektroniki przed wszystkimi. Dzięki niej możesz zbudować zaawansowany układ elektroniczny bez konieczności wykonywania czasochłonnych projektów i żmudnych obliczeń. Arduino błyskawicznie zyskało ogromną popularność, a w ślad za nią pojawiły się w sprzedaży liczne dodatkowe moduły. Pozwalają one zbudować dowolne urządzenie — ogranicza Cię tylko Twoja wyobraźnia!

Ta wyjątkowa książka została w całości poświęcona platformie Arduino. Znajdziesz tu szczegółowe omówienie jej możliwości, liczne przykłady oraz opisy. W trakcie lektury dowiesz się, jak przygotować środowisko pracy, oraz rozpoczniesz tworzenie prostych projektów korzystających z cyfrowych portów wejścia-wyjścia. W kolejnych rozdziałach poznasz coraz bardziej zaawansowane możliwości Arduino. Wykorzystasz silniki prądu stałego i serwomechanizmy, zastosujesz czujniki ultradźwiękowe oraz wyświetlisz informacje na wyświetlaczu LCD. Ponadto przekonasz się, że można zintegrować Arduino z systemem iOS oraz innym oprogramowaniem. Książka ta jest doskonałą lekturą dla wszystkich pasjonatów elektroniki.

DZIĘKI TEJ KSIĄŻCE:

- poznasz tajniki platformy Arduino
- wykorzystasz czujniki oraz silniki
- skomunikujesz się z układem za pomocą sieci
- odkryjesz dla siebie potencjał drzemiący w tej platformie!

Twoja przepustka do świata elektroniki!

helion.pl księgarnia internetowa

Nr katalogowy: 19327

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900 / 0 601 339900

Helion

Sprawdź najnowsze promocje: http://helion.pl/promocje
Książki najchętniej czytane: http://helion.pl/bestsellery
Zamów informacje o nowościach: http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN: 978-83-246-6356-9

9 788324 663569

Cena: 69,00 zł